

List of Figures and Tables

Figure 1. Explanation of Symbols in the Definition of Adaptation.....	5
Figure 2. Typical Display on the LCD of a Test Instrument.....	7
Figure 3. A Simplified Model of User Interface for Test Instruments	8
Figure 4. Software Architecture for User Interface.....	10
Figure 5. Symbols for Different Degrees of Satisficing.....	13
Figure 6. Usage Scenarios for SA3	15
Figure 7. The Functional Requirement for SA3.....	16
Figure 8. The Architecture for SA3.....	17
Figure 9. User Interface for SA3	19
Figure 10. Template for Adding NFR Decomposition Methods to KB	20
Figure 11. Graphical Output from SA3 that Displays Instances of NFR Softgoals	21
Figure 12. Software Architecture for User Interface.....	23
Figure 13. SIG For Connections	26
Figure 14. SIG For Patterns.....	27
Figure 15. SIG For Constraints	28
Figure 16. SIG For Styles	29
Figure 17. SIG For Rationales	30
Figure 18. Generated Architecture for Dynamic Cursor Rearrangement.....	33
Figure 19. SIG for Correlations.....	34
Figure 20. Generated Architecture for Dynamic Cursor Rearrangement with Correlation	34
Figure 21. Generated Architecture for Dynamic Field-Width Adjuster.....	35
Figure 22. DFD for Dynamic Field-Width Adjustment Feature.....	36
Figure 23. Architecture for Adaptable Screens.....	37
Figure 24. Popup For Adding New Parameter.....	38
Figure 25. Screen with new Parameter Added	39
Figure 26. Screen with Popup for Cursor Movement Rearrangement	40
Figure 27. Screen Display before Adaptation for Dynamic Field Width	41
Figure 28. Screen Display after Adaptation for Dynamic Field Width.....	42
Figure 29. Popup for Formula Selection	43
Figure 30. Template for Adding New Screen	44
Table 1. Illustration of KB Properties of NFR Framework	13
Table 2. Claim Softgoals in Figure 4	25

Adaptable User Interface Generation

Nary Subramanian
Firmware Engineer
Anritsu Company, Applied Technology Division
Richardson, TX 75081

Lawrence Chung
Dept. of Computer Science
University of Texas at Dallas
Richardson, TX 75081

Abstract

User Interface (UI) is that subset of a software system that interacts with the user of the system. Being a software system in itself, UI possesses certain attributes or non-functional requirements (NFRs) such as usability, reliability, simplicity, unambiguity, etc. However, recently, adaptability is emerging as an important characteristic for UI systems. Briefly, adaptability is the ability of a system to accommodate changes in its environment. As for any other software system, the first step in the development of a UI is the creation of the architecture for the system, and in order for the UI to be adaptable, the architecture of the UI should itself be adaptable. This paper focuses on semi-automatic generation of adaptable User Interfaces by using a tool called the Software Architecture Adaptability Assistant (or SA3). SA3 uses the principles behind the NFR Framework, particularly the latter's knowledge base properties, to automatically generate adaptable architectures, which can then be completed by the UI developer, if needed. In order to validate the architectures generated by the tool, we used the domain of embedded systems, in particular, test systems. SA3 generated adaptable architectures for UI for these systems and we implemented the architectures to confirm their adaptability.

1. Introduction

The user of a software system interacts with it through its User Interface (UI). The User Interface is in itself a software system and as such possesses attributes or non-functional requirements (NFRs) such as usability, reliability, simplicity, unambiguity, etc. [1]. However, adaptability is emerging as an important characteristic for UI systems [1¹, 2, 3, 4, 5, 6]. Adaptation of user interfaces has been a problem considered for a long time [24]. Among the studies done in [24] are self-adapting systems; another study uses adaptation for tailoring screens as per requirements [25]; while yet another reason for adaptation is the development of intelligent user interfaces [26].

Intuitively, adaptability can be defined as the ability of a system to accommodate changes in its environment. As for any other software system, the first step in the development of a UI is the creation of its architecture, and in order for the UI to be adaptable, its architecture should itself be adaptable [7, 8, 23]² ([23] says that the adaptation for user interfaces must be considered from the earliest stages of design).

But how do we develop adaptable architectures? We found that goal-oriented approaches are useful in this regard which treat adaptability as a goal to be achieved during the process of software development. The NFR Framework [9, 10, 11] facilitates goal-oriented development and we used this Framework to help develop adaptable architectures. One of the nice properties of the NFR Framework is that it permits easy implementation of its elements in a knowledge base, the consequence of this being that the developed adaptable architectures can be populated in a knowledge base and then searched for. This helps automatic architecture generation, which can then be completed, if needed, by the developer. In this paper we present the tool called the Software Architecture Adaptability Assistant (SA3) that helps generate adaptable architectures for user interfaces.

There have been several tools developed for this purpose in the literature: the Teallach tool [6] helps designers construct models and support the design of user interfaces by providing facilities for relating the

¹ See Chapter 13: Adaptive Design, pp. 291-325.

² See pages 15-17 in [7], page 32 in [8], and page 178 of [23]

different models; the Project SCOUT (Structured Documentation for User Interface Specifications) provides a framework for managing generic UI specifications as well as for automatic production of specifications [12]; XXL [13] is an environment for automated building of graphical user interfaces; TACTICS [14] is a tool that helps in the automatic generation of user interfaces along with design transformations; GIPSE [15] is yet another tool that helps in the generation of self-running applications; SAUCI (page 310 of [1]) is a tool that helps design adaptive user interface for UNIX-based systems; N-CHIME (page 315 of [1]) helps develop adaptive and adaptable user interface designs. In the field of software engineering there have been several tools developed and a brief overview can be seen in [16]. However, the tool that we developed in this paper, the SA3, helps develop adaptable architectures for user interfaces (preliminary versions of this paper appeared in [16,37]).

In order to validate the architectures generated by SA3 we implemented the architectures in a real embedded system, a test instrument. The resulting user interfaces were then confirmed to be adaptable. However, what do we mean by “adaptable”? A survey of literature [17] has shown that there is no single uniform definition of this NFR; almost each paper has its own definition. In order to be consistent we have given our definition of this NFR in this paper.

In this paper we have used UML [18] for design description although any other language with similar modeling power may also be used. Section 2 of this paper gives our definition of adaptability and introduces architectural concepts; Section 3 gives a brief overview of User Interfaces; Section 4 discusses the NFR Framework; Section 5 presents the tool, the SA3; Section 6 discusses the architecture generation for user interfaces by the tool; Section 7 presents the results of validation of the tool’s architectures; and Section 8 gives the conclusions.

2. Adaptability and Architecture

2. Adaptability and Architecture

In this section we review the concepts of adaptability and architecture. These concepts as described here will be used in the rest of the paper.

2.1 Adaptability

The definition of adaptability that we give below has been mentioned earlier [16,17]. Adaptation means change in the system to accommodate change in its environment. More specifically, adaptation of a software system (S) is caused by change (δ_E) from an old environment (E) to a new environment (E'), and results in a new system (S') that ideally meets the needs of its new environment (E'). Formally, adaptation can be viewed as a function:

$$\text{Adaptation: } E \times E' \times S \rightarrow S', \text{ where } \text{meet}(S', \text{need}(E')).$$

A system is adaptable if an adaptation function exists. Adaptability then refers to the ability of the system to make adaptation.

Adaptation involves three tasks:

1. ability to recognize δ_E
2. ability to determine the change δ_S to be made to the system S according to δ_E
3. ability to effect the change in order to generate the new system S' .

These can be written as functions in the following way:

$$\text{EnvChangeRecognition : } E' \times E \rightarrow \delta_E$$

$$\text{SysChangeRecognition : } \delta_E \times S \rightarrow \delta_S$$

SysChange : $\delta_S x S \rightarrow S'$, where $meet(S', need(E'))$.

The *meet* function above involves the two tasks of validation and verification, which confirm that the changed system (S') indeed meets the needs of the changed environment (E'). The predicate *meet* is intended to take the notion of goal *satisficing* of the NFR Framework [9,10,11], which assumes that development decisions usually contribute only partially (or against) a particular goal, rarely “accomplishing” or “satisfying” goals in a clear-cut sense. Consequently generated software is expected to satisfy NFRs within acceptable limits, rather than absolutely. In the above δ_E is the difference between E' and E .

Figure 1 explains the relationship between the various symbols described above.

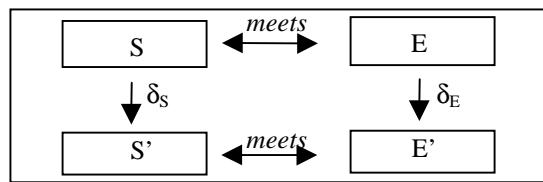


Figure 1. Explanation of Symbols in the Definition of Adaptation

2.2 Architecture

Any architecture usually consists of the following constituents: components, connections, patterns, constraints, styles, and rationales [7,8]. Components are the elements from which systems are built; connections are the interactions between the elements; patterns describe the layout of the components and connections; constraints are on the components, connections and patterns; styles are an abstraction of architectural components from various architectures; and rationales describe why the particular architecture was chosen. Thus, for example, Interrupt Handler and Parser could be components, message passing could be the connection between them, style could be layered, constraint could be that interrupt handler should accept all interrupts and that the data received from the interrupt handler should be sent to the parser within 100ms (constraint on connection), pattern could be sequential processing, and rationale could be familiarity with this architecture.

3. User Interfaces

This section gives a brief overview of user interfaces as applicable to the test instrument industry. User interfaces help users of the software system interface with the software system. User interfaces are usually firmware, including hardware and software aspects. Thus there is keyboard or mouse which the user uses to select or enter data representing the hardware part of the interface and the software application running in the PC that detects these keyboard/mouse signals and takes appropriate actions representing the software part of the user interface. Usually there are a variety of hardware devices for an user to interface with a software system: keyboard, front panel keys, knobs, joystick, LCD displays, serial ports, parallel ports, monitors, touch-sensitive screens, audio inputs/outputs, etc. Many of these inputs/outputs are usually found in embedded systems. This is true of test instrument industry as well, since many of the test instruments are themselves embedded systems. The test instruments are used for a variety of purposes such as measuring power, frequency, digital signals, protocols, etc (see [19,20] for a list of various test instruments). There are advantages and disadvantages when designing user interfaces for test instruments. Advantages occur because the range of interactions can be constrained – limited choices may be provided to the user; disadvantages occur because customers decide on the product's quality based on the user interface the product provides. This means in order for the test instrument companies to boost their profits user interfaces play a vital role, even though the interactions may be limited (for example, usually test instruments do not provide a large display with a keyboard and mouse; a front panel, a knob, a small LCD display are all that usually exist).

A typical display on the LCD of a test instrument is shown in Figure 2. There are twelve different parts to the display of Figure 2 and they are described below:

1. Name of the test instrument which is indicated in Figure 2 as “Test Instrument”
2. Name of the display, usually referred to as “screens”, which is “STANDARD SCREEN” in Figure 2
3. Four different Input sections: the “Call Processing Mode”, “Frequency”, “Level” and “Tests” – the inputs to the instrument are indicated by square braces ([...]).
4. The output from the instrument indicated by “Call Proc: Stop” – these are fields updated by the instrument on its own
5. The calculated value based on the inputs indicated by “Total Output Level” and by normal braces
6. The cursor, which is on the “In Call” parameter and which can be moved around by cursor keys
7. The vertical soft keys indicated by “VKey1” to “VKey6”
8. The horizontal soft keys indicated by “HKey1” to “HKey5”
9. The vertical page indicator – which indicates the number of vertical pages, shown by the “2” under vertical keys
10. The horizontal page indicator – which indicates the number of horizontal pages, shown by the “1” next to “HKey5”

11. Special button “Setup Instrument” than is used for common instrument setup tasks, such as time setting, etc.
12. Color contrast for the various parts (though Figure 2 is in black and white).

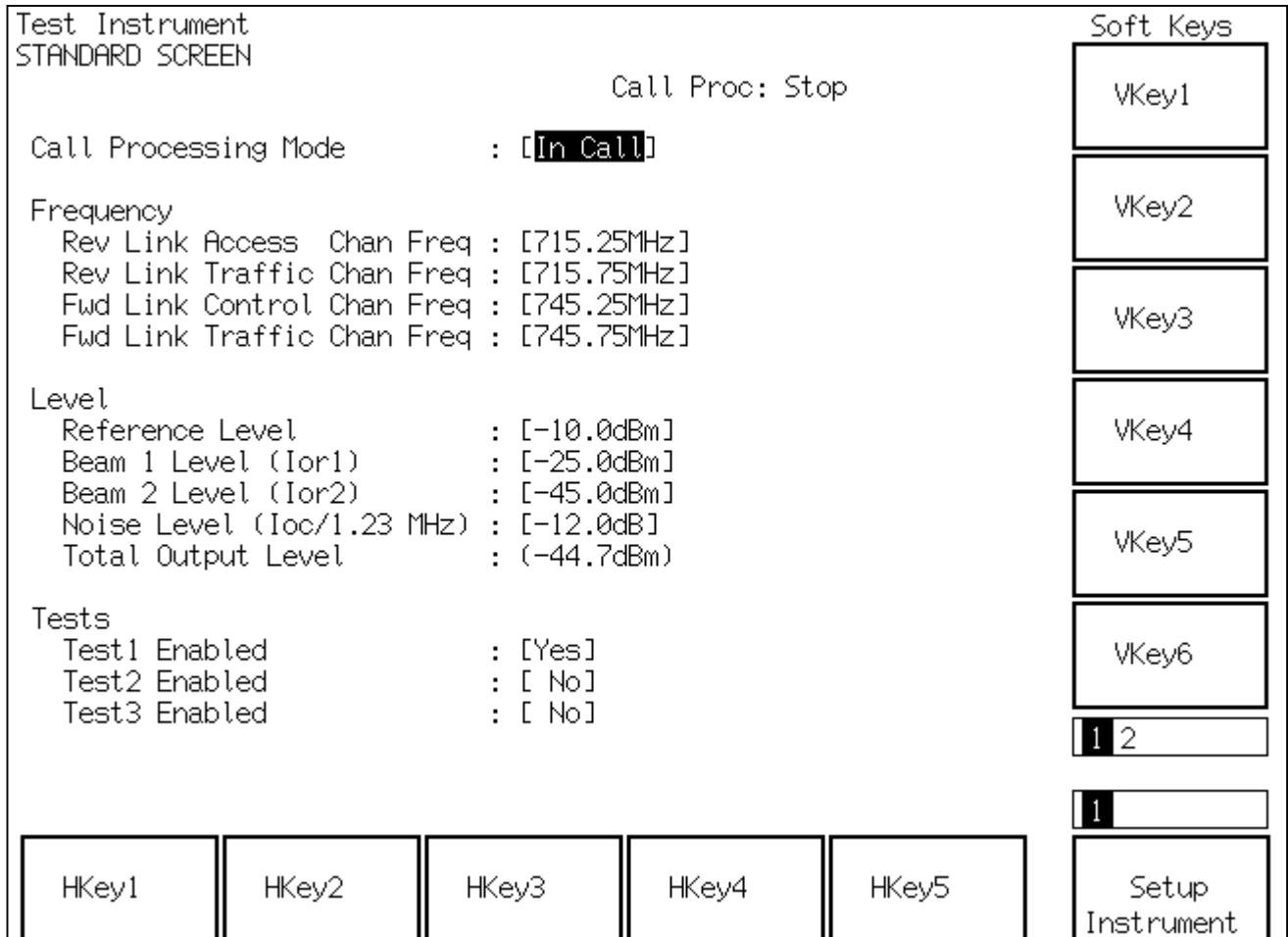


Figure 2. Typical Display on the LCD of a Test Instrument

Thus it can be seen that screens in an instrument are used for both – input and output of information. In this paper we will concentrate on the design of these screens and further how the screens can adapt to environment changes.

3.1 User Interface Adaptability

What does it mean to say user interface is adaptable? As mentioned in our definition of adaptability, an adaptable user interface can accomplish the following tasks:

- Detect changes in environment
- Recognize need for system change
- Change the system

The above tasks may be accomplished in some way or the other: automatically or manually, proactively or reactively, continuously or at discrete points in time, and so on. Therefore in order to answer the question posed earlier we need to first determine the environment for the screen of a test instrument and to help us in this task a simplified model of typical user interface in test instruments is given in Figure 3.

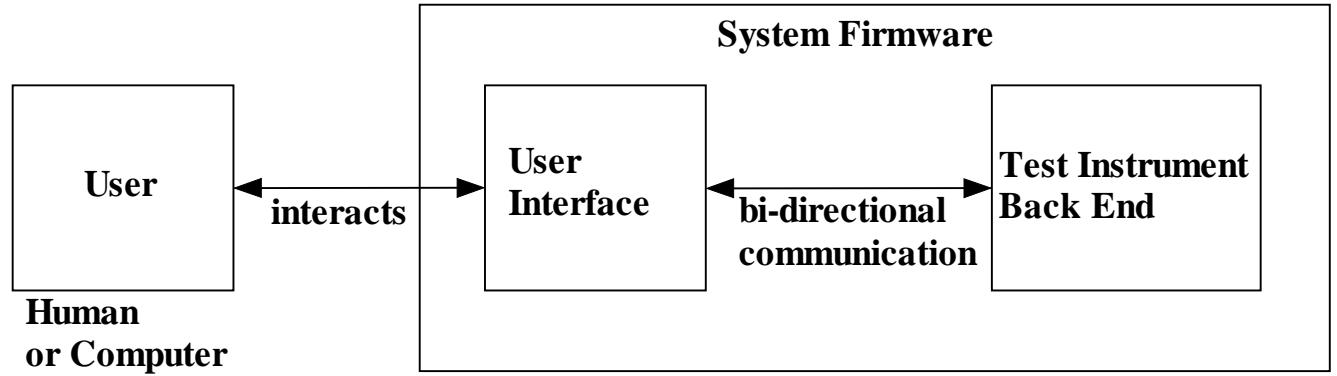


Figure 3. A Simplified Model of User Interface for Test Instruments

Figure 3 is similar to those in [1,7]³. The user interacts with the system firmware through the system's user interface. The user could be a human pressing the front panel keys or a computer running an automated script and interacting through serial/parallel connections. The user interface is the front end of the system and communicates with the back end for the processing of user's inputs and for transmitting information to the user. Thus the environment for the user interface consists at least of:

1. Front panel keys/knobs
2. Communication ports (serial/parallel ports)
3. Display (related to human user's perception)
4. Test instrument back end

Thus the environment could change along any one of at least these four variables. An adaptable user interface will adapt to these environmental changes. And for the particular aspect of user interface considered in this paper, viz., the screens for a test instrument, the adaptation can occur along any one of the twelve parts discussed earlier.

3.2 Architecture of User Interface

Based on the model of Figure 3, the software architecture for the user interface is shown in Figure 4. The architecture is shown in layered style and has four basic layers: the hardware interface layer, the basic tasks layer, the display control layer and the back end control layer. The hardware interface layer provides the interface to the user-interface hardware and usually has drivers for the various physical interfaces supported by the test instrument. The basic tasks layer schedules the interrupts received from the physical world, stores parameters which are used in the display and provides a guard to the display so that at one time the display is writing only one thing⁴. The display control layer handles all display related events including drawing screens, updating/retrieving parameters displayed on the screen, drawing the cursor, popping-up windows for error messages, help messages, etc., and for updating on the screen information received from the back end as well as any calculated parameters. The back end control layer(s) handle formatting/unformatting messages to/from the back end and the actual communication with the back end. Now that we have the architecture, how does the architecture help with adaptability? As mentioned in the Introduction, the user interface will be adaptable provided its architecture is adaptable. And this is possible by adapting one or more architectural constituents: components, connections, patterns, constraints, styles

³ See figure on page 39 of [1] and Figure 5.2 on page 101 of [7]

⁴ For a discussion on hardware-software interfaces please refer to [21,22].

and rationale. There are several mechanisms to perform adaptation and we will discuss them in the next section.

3.3 Mechanisms for Adaptability of Architecture of User Interface

In Figure 4 the style of the architecture is layered which means that only adjacent layers can communicate with each other. Thus if the user interface should be adaptable to the environment change: “back end communications rate exceeds the user interaction rate”, then perhaps shared data style will be more appropriate to help back end data be displayed on the screen. This is an example of style adaptation. Perhaps for this adaptation manual intervention may be required – the system may have to be redesigned and reimplemented for this adaptation. But there are many other cases of adaptation where less manual intervention may suffice. Thus we can have the following mechanisms of adaptation (the following list is not meant to be exhaustive):

1. Component adaptation
 - Components with multiple capabilities – thus a component could display both English and Japanese characters
 - Components that can specialize the function of general components – these components form an inheritance mechanism and a controller selects the appropriate component perhaps at run-time
 - Components that use rules for their functioning – these rules may be changed for adaptation.
 - Changing components
2. Connection adaptation
 - Using mediator objects for connections – such objects can change the interconnections between components without the components being aware of this
 - Changing interconnections
 - Providing a library of differently interconnected components so that one of them may be used as needed
3. Pattern adaptation
 - Changing patterns
 - Using a library of different patterns of components and their interconnections so that the needed one may be used
4. Constraint adaptation
 - Changing (adding, deleting, modifying) constraints on components
 - Changing (adding, deleting, modifying) constraints on connections
 - Changing (adding, deleting, modifying) constraints on patterns
5. Style adaptation
 - Using a repository of architectures of different styles each one to be used in a specific situation
 - Changing styles
6. Rationale adaptation
 - Changing rationale for the architectures – compromising/satisfying more some other NFRs for the sake of adaptability
 - Providing a repository of different architectures with different rationales to be used in specific situations
7. Combination of one or more of the above.

These mechanisms will be revisited after we have introduced the NFR Framework in the next section.

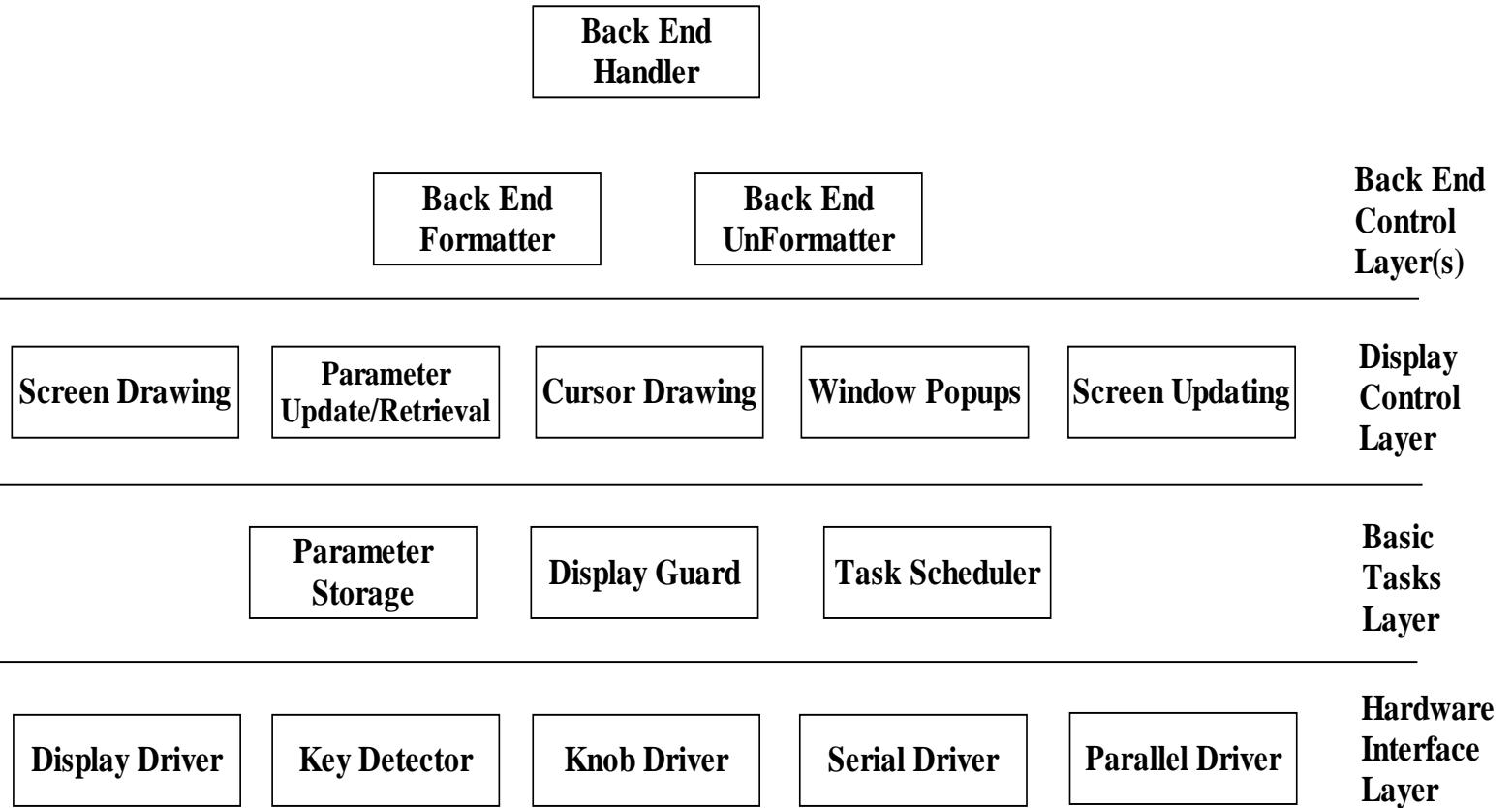


Figure 4. Software Architecture for User Interface

3.4 The Requirements for User Interface

The previous discussion gives a somewhat detailed feel for the user interface of a system. For the purposes of further discussion in this paper we give the detailed requirements that the User Interface should meet. The rest of the paper focuses on generating architectures to meet these requirements.

3.4.1 System Requirements for User Interface

1. There are keys for alpha-numeric data entry
2. There are up/down/left/right keys for cursor movement
3. There are 6 vertical soft-keys and 5 horizontal soft-keys for special functions
4. There is an LCD Display
5. There is a mono-tone audio output (the “beep”)
6. The back end communication messages are detailed in the Back End Interface Document⁵

3.4.2 Functional Requirements for User Interface Software

1. The user interface shall respond to all key presses
2. The user interface shall respond to all soft-key presses
3. The user interface shall display the STANDARD SCREEN (Figure 2)
4. The user interface shall beep upon pressing of any soft-key

⁵ This Back End Interface Document is mentioned here only for informational purposes; this document is not relevant to the subsequent discussions in this paper.

5. The range of values for all the parameters in STANDARD SCREEN are given in the Data Dictionary⁶
6. The Total Output Level value formula is also given in the Data Dictionary
7. The back end updates should be displayed next to “Call Proc:” in the STANDARD SCREEN
8. Error messages should be displayed for any incorrect key presses in a separate pop-up window saying “Error”

3.4.3 Non-Functional Requirements for User Interface Software

1. The software should be adaptable to the following environmental changes:
 - Should permit range of values for parameters to be changed
 - Should permit new screens to be added
 - Should permit the calculated values to obey new formulas
 - Should permit addition of new parameter/value/update fields
 - Should permit color changes for the various displays
 - Should permit the cursor movement to be rearranged
2. The software should not miss any key press or back end data
3. The software should have fast response time.

4. The NFR Framework

Consideration of NFRs during the process of architecture development requires systematic methodology. NFRs tend to interact with each other either synergistically or in conflict. A systematic approach to analyzing these interactions between NFRs is required to develop an architecture that satisfies various NFRs. ATAM [29] is one such method; another is the estimation method [31]; yet another method is ALMA [30]. However, in our opinion none of these methods help to consider NFRs during the process of software development (as opposed to just the design phase). The NFR Framework [9,10,11] helps to continuously monitor the impact of various decisions during the process of software development.

The NFR Framework requires the following interleaving tasks, which are iterative:

1. Develop the NFR goals and their decomposition.
2. Develop architectural alternatives.
3. Develop design tradeoffs and rationale.
4. Develop goal criticalities.
5. Evaluation and Selection.

The NFR Framework has a defined ontology that helps to depict NFRs, design constituents, claims, and relationships. During the application of the above five steps, a diagram is created that relates the NFRs and design constituents. This diagram is called the softgoal interdependency graph or SIG. An example SIG is given in Figure 12. The various elements of that SIG are described below.

4.1 Softgoals

In the NFR Framework each NFR is called an NFR softgoal, each design constituent is called a design softgoal, while a claim is called the claim softgoal. A claim justifies an element of the NFR Framework. All softgoals are depicted by clouds – the NFR softgoal is depicted by normal clouds, the design softgoal by dark clouds and claims by dashed clouds. All softgoals are named in the following convention:

Type[Topic1, Topic2, ...],

⁶ Again the Data Dictionary is mentioned for informational purposes only; in subsequent discussions any relevant material from this dictionary will be explicitly mentioned.

where *Type* is an NFR and *Topic* is a system to which the *Type* applies. Thus in Figure 12, *Adaptability[UI]* is an NFR softgoal of type *Adaptability* and with topic UI. UI stands for User Interface. Likewise *Cursor Movement Matrix* is a design softgoal while *Claim1* is a claim softgoal. The design constituent corresponding to a design softgoal could be any one of the architectural constituents: components, connections, patterns, constraints, styles and rationale (Section 2.2).

Softgoals can also have priorities or criticalities – this is the importance of the softgoals. The criticalities are indicated by ‘!’ marks. Criticalities are assigned during Step 4 of the NFR Framework.

4.2 Contributions

Another feature of SIGs are the contributions of child softgoals to its parents. Contributions are depicted by arcs and lines. There are several types of contributions: equal (only one child), AND contribution (indicated by a single arc), OR contribution (indicated by double arc), MAKE contribution (green line), HELP contribution (blue line), HURT contribution (orange line) and BREAK contribution (red line). One of the main features of the NFR Framework is the decomposition of softgoals of all types.

One may wonder how one goes about decomposing softgoals. This is entirely dependent on the application domain. And there can be several decompositions for an application domain. The decomposition shown in Figure 12 is only one of the possible decompositions for the UI domain. Several other decompositions will be shown later.

It may be noted that the MAKE, HELP, HURT and BREAK contributions actually indicate the different degrees of satisfying as shown by the color code of Figure 5. Different architectures (due to Step 2 of the NFR Framework) may generate different design softgoals and these design softgoals may satisfy the NFR softgoals differently. This lets us compare the architectures with respect to their quality attributes (Step 5 of the NFR Framework). The contributions are decided during Step 3 of the NFR Framework and the rationales result in the claim softgoals.

4.3 Correlations

One of the features of NFRs is that they interact synergistically or in conflict. Thus while one design softgoal may HELP one NFR softgoal, the same design softgoal may break another NFR softgoal. These type of conflicts are depicted using correlations – they are shown by dashed lines in SIGs.

4.4 Knowledge Base Properties

It was mentioned in the introduction that the NFR Framework is knowledge base friendly. In this section we will show frame-like notations for the various elements of the NFR Framework. The frame-like notations can be used to populate a knowledge base with the elements of the Framework. A discussion of these properties can be seen in [16,17]. However, a brief description of the knowledge base features is given in Table 1.

Thus a knowledge base populated with various elements of NFR Framework having the following properties:

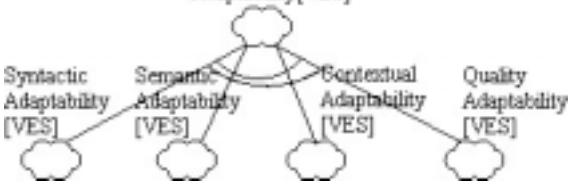
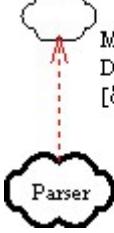
1. all NFR softgoals are adaptability-related, and
2. design softgoals satisfy these NFR softgoals,

could be searched for design elements (corresponding to design softgoals) based on the NFR softgoals.

Strongly Positive Satisficing or MAKE
Positive Satisficing or HELP
Negative Satisficing or HURT
Strongly Negative Satisficing or BREAK

Figure 5. Symbols for Different Degrees of Satisficing

Table 1. Illustration of KB Properties of NFR Framework

NFR Framework Element	Knowledge Base Description
Adaptability[VES] 	NFRSoftgoal AdaptabilityVES Type: Adaptability Topic: VES Label: Undecided Priority: Low Author: Nary Creation Time: 27 March 2002
Adaptability[VES] 	NFR DecompositionMethod VESAdaptabilityViaSubType Parent: Adaptability[VES] Offspring: Syntactic Adaptability[VES], Semantic Adaptability[VES], Contextual Adaptability[VES], Quality Adaptability[VES] Contribution: OR
Manual Detectability [δ_E] 	CorrelationRule ParserBREAKSManualDetectability Parent: Manual Detectability[δ_E] Offspring: Parser Contribution: BREAKS Condition: Parser cannot manually detect δ_E

This will help to develop adaptable architectures semi-automatically at least. Examples of such systems in fields other than software can be found in [32,33]. A tool for generating adaptable architectures based on these ideas has been developed, called the Software Architecture Adaptability Assistant (or SA3), and is described in Section 5.

4.5 The User Interface and the NFR Framework

In this section we show how the discussions in Section 3 fits in with the NFR Framework. Firstly based on the functional requirements the architectural constituents – components, connections, patterns, constraints, styles and rationales – are created. Each of these constituents becomes a design softgoal. The various non-functional requirements become NFR softgoals and these NFR softgoals are decomposed into their relevant sub-NFR softgoals. The various design softgoals satisfy the various NFR softgoals to greater or lesser extent and these are indicated by operationalization methods. The mechanisms for adaptability discussed in Section 3.3 also turn into operationalization methods. The rationale for the satisfying of the NFR softgoals by the design softgoals are captured by claim softgoals in the form of argumentation templates. Finally some design softgoals have correlations with more than one type of NFR softgoal and these correlations are captured by correlation rules. In the subsequent sections we will be seeing these terms referred to more often.

5. The SA3 Tool

In this section we describe the tool – the Software Architecture Adaptability Assistant or SA3 – that we developed to (semi-)automatically generate adaptable architectures for embedded systems. We first explain the principles behind the tool, then give the requirements for the tool, the architecture for the tool and describe its implementation including some screen shots. In the next section we explain how the tool can be used to generate architectures.

5.1 Principles behind SA3

In this section we will be tying together the concepts presented earlier. The starting point for the development of SA3 is the fact that NFR Framework permits creation of catalogs (Section 3.4). Catalogs of various architectural constituents (Section 2.2) that includes components, connections, patterns, constraints, styles and rationales can be created. However, on what basis are these catalogs created? This is where the functional requirements come in, and more specifically application domains enter the picture. Catalogs can be created for various application domains – the functional requirements for the application domains help generate various constituents of the architectures for the domain. Each of the architectural constituents though satisfying specific functional requirements for the domain, satisfies (Section 2.1) various non-functional requirements for the domain differently. The satisfying of the various NFRs for the domain is determined using the NFR Framework. The result of this is that we can create a knowledge base of the various architectural constituents. Subsequently, whenever an architecture for the domain satisfying known NFRs has to be generated, the knowledge base is searched for the appropriate architectural constituents. These constituents form the starting point for developing a complete architecture (in many cases the generated architecture may be sufficient). Thus the SA3 helps the designer create architectures for a domain quickly and efficiently by reusing constituents and the knowledge of the domain. However, this concept of reuse is different from that found in literature on reuse [28,29] in that here reusable constituents of the architecture are determined on the basis of the NFRs for the system being developed (and not on the basis of functional requirements, which is usually the case). This is similar in principle to the Hypothetical Architect's Assistant [33]. As the population of the knowledge base increases, the variety of architectures that can be automatically developed increases as well.

1. Populate the KB (Knowledge Base) with Adaptability related NFR softgoals
2. Populate the KB with the design softgoals based on the functional requirements for the domain of interest - in this case the VES domain; the functional requirements dictate the various architectural constituents created
3. Populate the KB with Decomposition Methods for the NFR Softgoals and design softgoals
4. Populate the KB with Correlations between design softgoals and various NFRs
5. Choose the starting softgoal for satisficing which an adaptable architecture has to be generated
6. Choose the type of architectural constituent to be searched for - whether it is for components, connections, patterns, constraints, styles or rationales
7. Choose the extent of satisficing to be searched for - whether strongly positively satisficing or positively satisficing
8. Based on the starting softgoal, the satisficing type and the constituent type, the SA3 checks if any decomposition methods for that starting softgoal are available
 - 8a. if so the tool does a depth first search to detect if any satisficing design softgoals of the constituent type are available
 - 8aa. if any design softgoal is available, the tool displays the architectural constituents corresponding to that design softgoal - all such satisfactory constituents are displayed sequentially
 - 8ab. if no design softgoal is available, SA3 displays an error message: "No architectural constituent of the required type available in the KB"
 - 8b. if no decomposition methods are available and the starting softgoal is not a design softgoal, then the tool displays an error message: "No architectural constituent of the required type available in the KB"
 - 8c. if no decomposition methods are available and the starting softgoal is a design softgoal then the constituent based on that design softgoal is displayed as the available adaptable constituent
9. By searching for all the different architectural constituents one after the other, the developer gets the starting point for the architecture. Either the generated architectural constituents will be sufficient or the developer can manually complete the architecture.

Figure 6. Usage Scenarios for SA3

In this paper the domain is that of the User Interface. In the NFR Framework each instance of an architectural constituent is a design softgoal. Thus by developing SIGs for each architectural constituent for the domain of User Interface and populating a knowledge base with these SIGs a readily usable store for architectural constituents is available. A search through this knowledge base gives the various architectural constituents satisfying a given NFR. In this paper all the NFRs are adaptability related. Hence the architectures generated by SA3 are adaptable as well. Figure 6 gives the usage scenarios for the SA3.

5.2 Requirements for SA3

The functional requirements for SA3 is shown succinctly by Figure 7. The input to SA3 is the adaptability related NFR softgoals that a system (S) is expected to satisfy, and the output from SA3 is a set of architecture constituents for system S that satisfies the input NFR softgoals. The system S belongs to the domain of VES.

The non-functional requirements for SA3 include that it should in itself be adaptable and easy to use.

5.3 Design of SA3

The functional requirements for SA3 can be satisfied in many different ways; however, the following components are a must:

1. a store for various adaptability-related NFR softgoals
2. a store for various design softgoals satisfying the functional requirements for the domain of VES.
3. a store for the interdependencies and correlations between the NFR softgoals and the design softgoals
4. a search facility to search the contents of the store
5. a user interface for populating the store and displaying results

For the “store” we had a choice of using either a DBMS or a KBMS (Knowledge Base Management System). Due to the advantages of a KBMS [34] such as deductive reasoning and powerful query capabilities, we decided to use a KBMS. An easy-to-use user interface is needed to populate the knowledge base, and to query the knowledge base for finding architectural constituents.

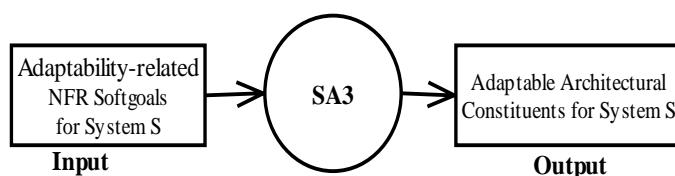


Figure 7. The Functional Requirement for SA3

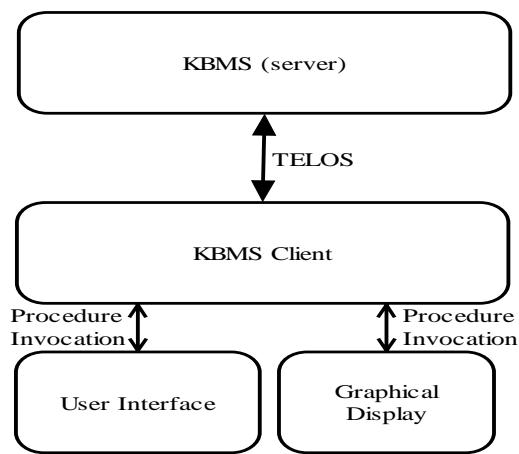


Figure 8. The Architecture for SA3

5.3.1 Software Architecture for SA3

When developing the architecture for SA3 we had several choices in terms of the architectural constituents such as components, connections, styles, constraints, etc.: the KBMS could be developed from scratch or a readily available KBMS could be used; the connections could be message passing or remote method invocation; the style could be object oriented or client server based. We decided to use a readily available KBMS called ConceptBase [36]; this choice of a component pretty much restricted the style to client-server and the interactions between the client and the server to X-window protocol. The architecture for SA3 is given in Figure 8. The language used to communicate with the KBMS is called Telos [35] – Telos permits advanced query capabilities and deductive reasoning. The platform for the KBMS is the UNIX operating system Solaris 2.4 or higher. The client ran on a Windows based system. The Tcl/Tk language was used to implement the client.

In Figure 8, the KBMS (server) resides on the UNIX server; the KBMS Client is on the Windows client. The KBMS Client communicates with the server using the Telos language. There are two other main components on the client side: the User Interface and the Graphical Display. The user interface is used to populate the KB and query the KB; the graphical display shows the instances of various elements of the NFR Framework. The generated architectural constituents are also displayed.

5.4 The User Interface for SA3

The SA3 User Interface is shown in Figure 9. The various parts of the user interface are highlighted in that figure. The Menubar contains the various menus of SA3. The available menus are Server, Edit, Browsing, Populate KB, Generate Architecture. Each of these menus has various sub-menus (the relevant ones will be explained later). The Icon Panel provides icons for handling the Editor Field – they include the knowledge base query operations such as Tell, Untell, Ask, Frame , and common operations such as Cut, Copy and Paste. Icons for Next and Back are also included (for scrolling in the Editor Field). The Editor Field lets the user create objects and then add to knowledge base or query objects in knowledge base. The Editor Field is not required for using SA3; this is kept there only for advanced users. The Protocol Field displays all communication between the user interface and the knowledge base (the server), including error conditions. The Status Line currently displays only one item – the status of connection with the server: if connected to the server, “connected” is displayed, else “disconnected” is displayed.

One of the sub-menus for Server is the “Connect CB Server ...” - this helps the client to connect itself to the server. Upon clicking on this sub-menu, a connection window pops up that allows one to connect to the server.

The various sub-menus of Populate KB item of the Menubar help the user to populate the knowledge base. The following sub-menus are currently available: “Add NFR Softgoal ...”, “Add Operationalizing Softgoal ...”, “Add NFR Decomposition Method ...”, “Add Operationalization Method ...”, and “Add Correlation Rule ...”. Each of these sub-menus allow, respectively, to add an NFR softgoal, to add a design softgoal for one of six architectural constituents – components, connections, patterns, constraints, styles and rationales, to add an NFR decomposition method, to add an operationalization method, and to add a correlation rule. Thus upon clicking “Add NFR Decomposition Method ...” sub-menu of the Populate KB item of the Menubar, the window as in Figure 10 pops up. Using the template of Figure 10, several NFR decomposition methods can be added to the knowledge base. Several such templates are provided to populate the knowledge base. The sub-menus for the Generate Architecture item of the Menubar are “Component ...”, “Connection ...”, “Pattern ...”, “Constraint ...”, “Style ...”, and “Rationale ...”. These sub-menus are used to generate architectures and their usage is explained later. The browsing feature of the user interface provides facilities to view instances of any element of the NFR Framework. Figure 11 shows one such graphical output from the tool that displays the various NFR softgoals instantiated.

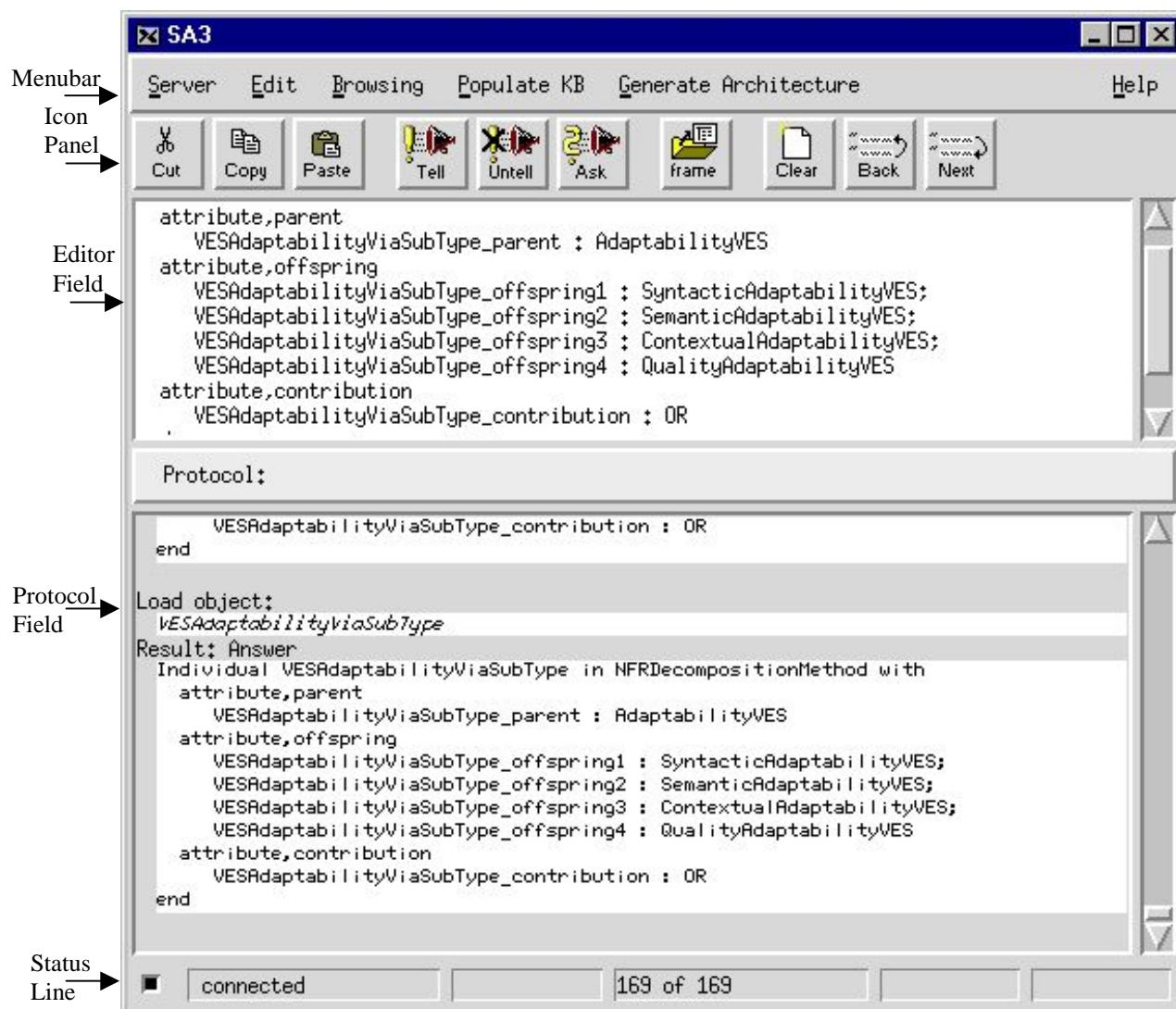


Figure 9. User Interface for SA3

Create NFR Decomposition Method

NFR Decomposition Method Name	:	VESAdaptabilityViaSubType
Parent NFR Softgoal	:	AdaptabilityVES
Offspring1 NFR Softgoal	:	SyntacticAdaptabilityVES
Offspring2 NFR Softgoal	:	SemanticAdaptabilityVES
Offspring3 NFR Softgoal	:	ContextualAdaptabilityVES
Offspring4 NFR Softgoal	:	QualityAdaptabilityVES
Offspring5 NFR Softgoal	:	
Offspring6 NFR Softgoal	:	
Offspring7 NFR Softgoal	:	
Offspring8 NFR Softgoal	:	
Offspring9 NFR Softgoal	:	
Offspring10 NFR Softgoal	:	
Contribution	:	OR

Create **Cancel**

Figure 10. Template for Adding NFR Decomposition Methods to KB

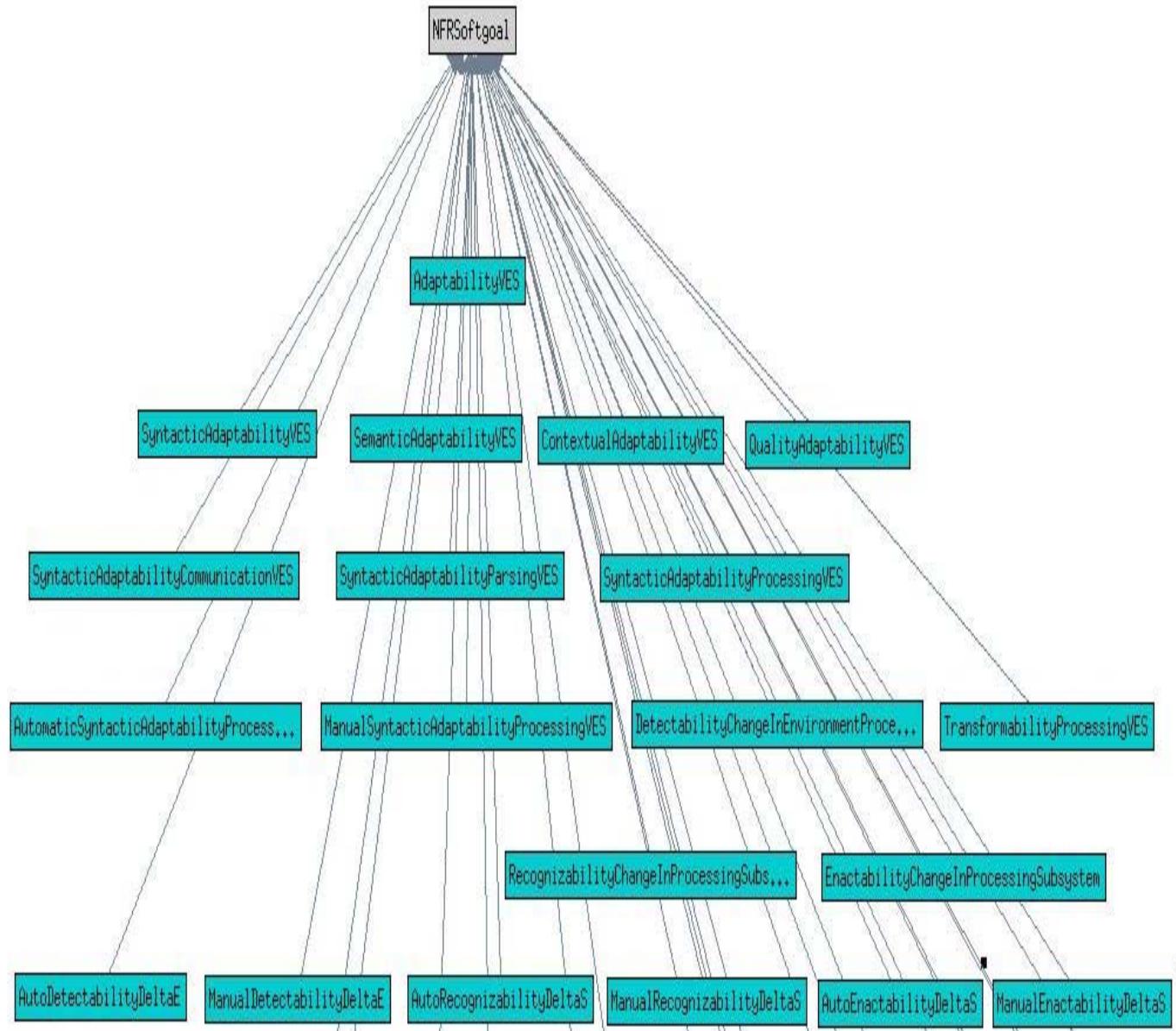


Figure 11. Graphical Output from SA3 that Displays Instances of NFR Softgoals

6. Populating the Knowledge Base

In this section we show how the knowledge base is populated with different elements of the NFR Framework for the user interface domain. For each constituent, a SIG is drawn and the reason for the SIG is also explained.

6.1 Populating the KB with Components

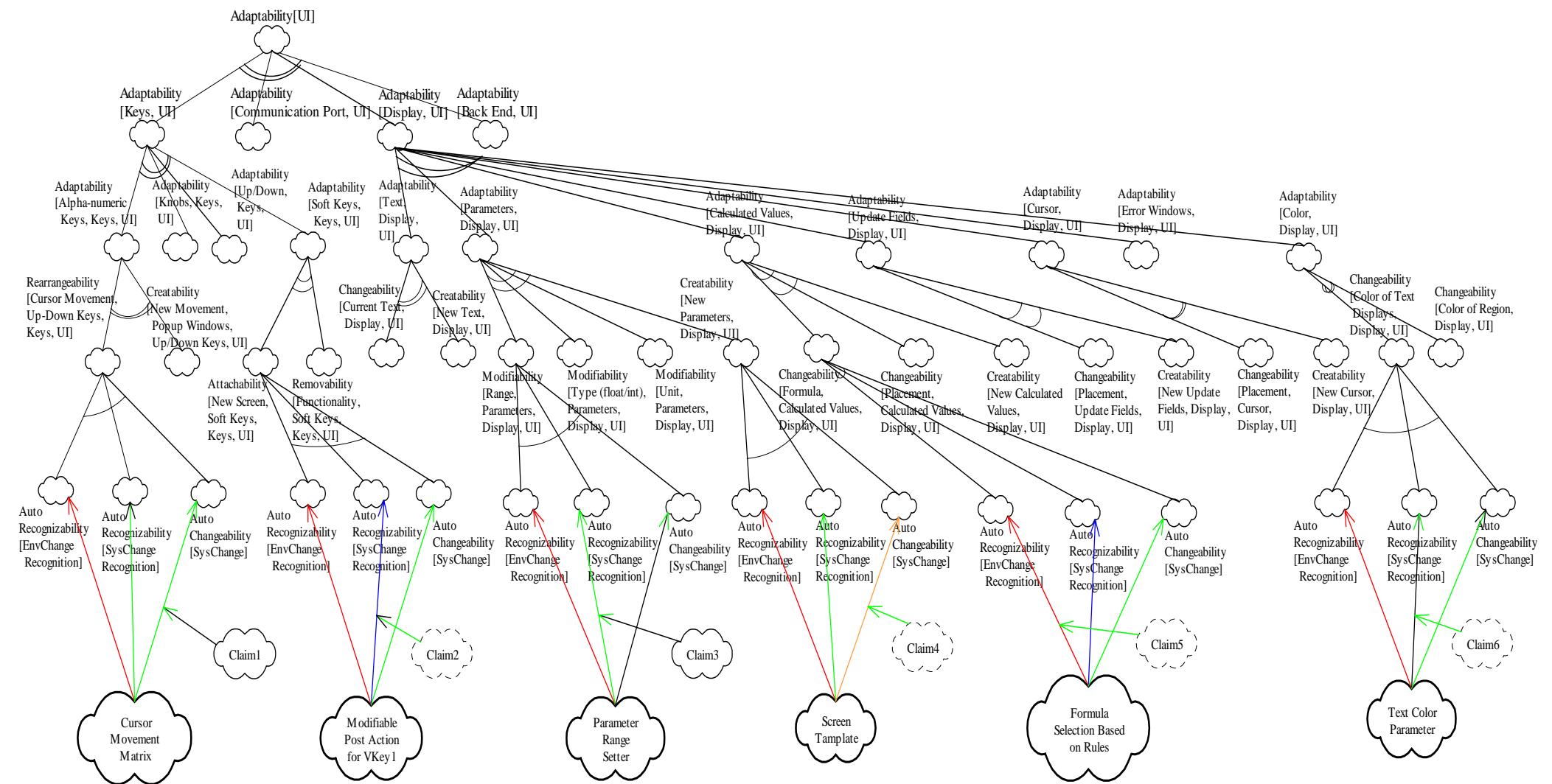


Figure 12. Software Architecture for User Interface

Figure 12 shows a SIG for some components. Each of the components helps in meeting some NFR for the target system. Thus Cursor Movement Matrix component helps in rearranging the cursor movement on the screen for the Up/Down/Left/Right keys; the Modifiable Post Action for VKey1 helps in attaching a new screen to VKey1; Parameter Range Setter helps in changing the range of parameters; Screen Template helps create new screens which could be attached to a soft key using Modifiable Post Action for VKey1; Formula Selection Based on Rules help change formulas for calculated fields; and Text Color Parameter help change the color of text displayed. The claim softgoals in Figure 12 are described in Table 2 below.

Table 2. Claim Softgoals in Figure 4

Claim	Description
Claim1	Once the cursor movement matrix is changed, the system subsequently changes the cursor movements automatically.
Claim2	If the modifiable post action for key1 has a mechanism to check for some changed environmental conditions then it can to some extent automatically recognize need for system change.
Claim3	Parameter Range Setter automatically recognizes the need for system change once the range of the parameters have been changed.
Claim4	Screen Template does not help in automatic system change; this facility may be provided to a limited extent though.
Claim5	Formula Based Selection of Rules component cannot automatically recognize environment change; the user has to choose the new formula based on a environment change (hence manual only).
Claim6	Once the Text Color Parameter's value has been changed it knows that there is a need for system change automatically.

The elements of the SIG of Figure 4 can be populated into the KB using the SA3 tool.

6.2 Populating the KB with Connections

The SIG for three different types of connectors is shown in Figure 13. The claim softgoals have been omitted for conciseness sake. Figure 13 shows how the various connectors satisfy adaptability-related NFR softgoals.

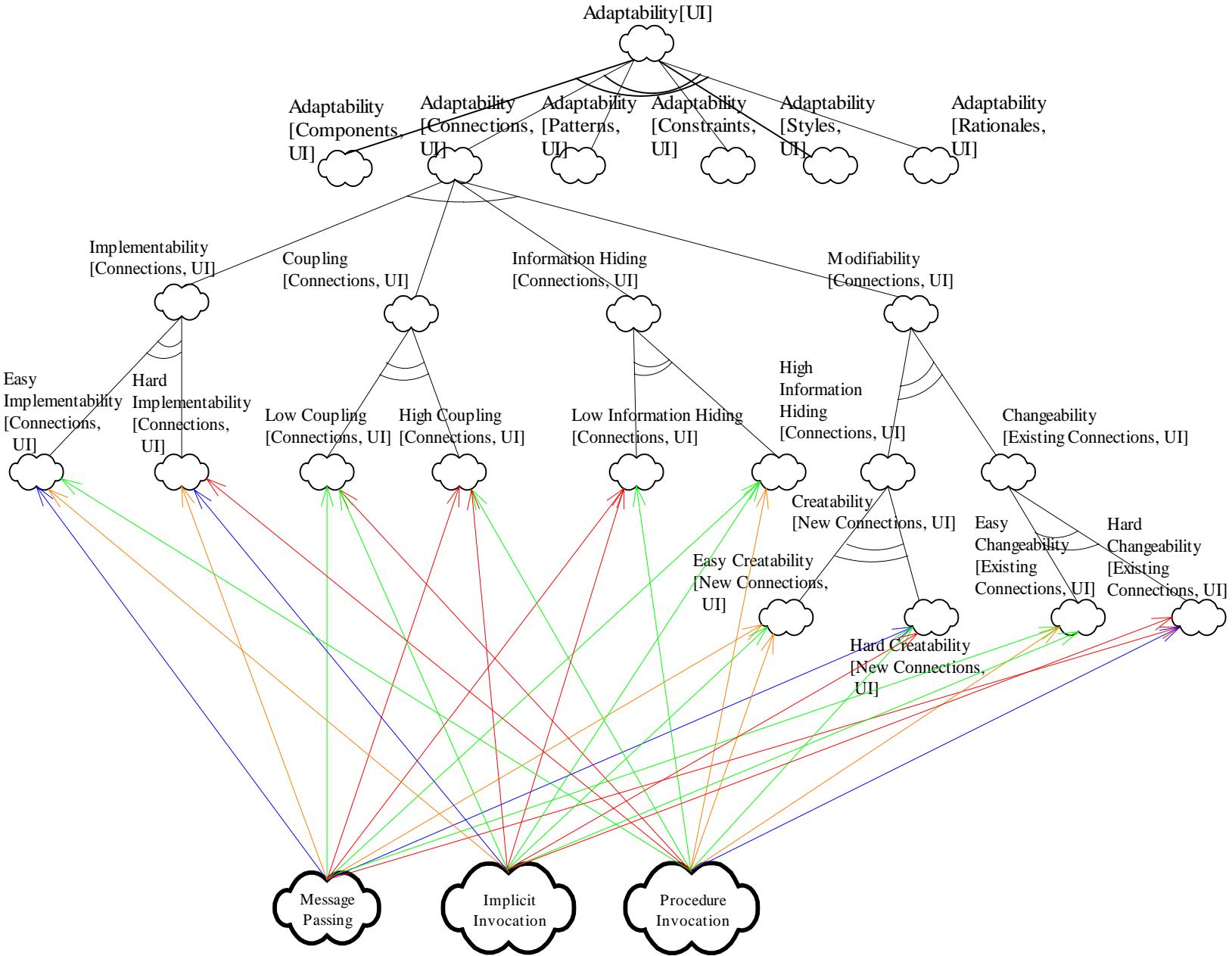


Figure 13. SIG For Connections

6.3 Populating the KB with Patterns

Figure 14 shows the SIG for three types of patterns: ring, sequential and star, and how these patterns satisfy adaptability-related softgoals.

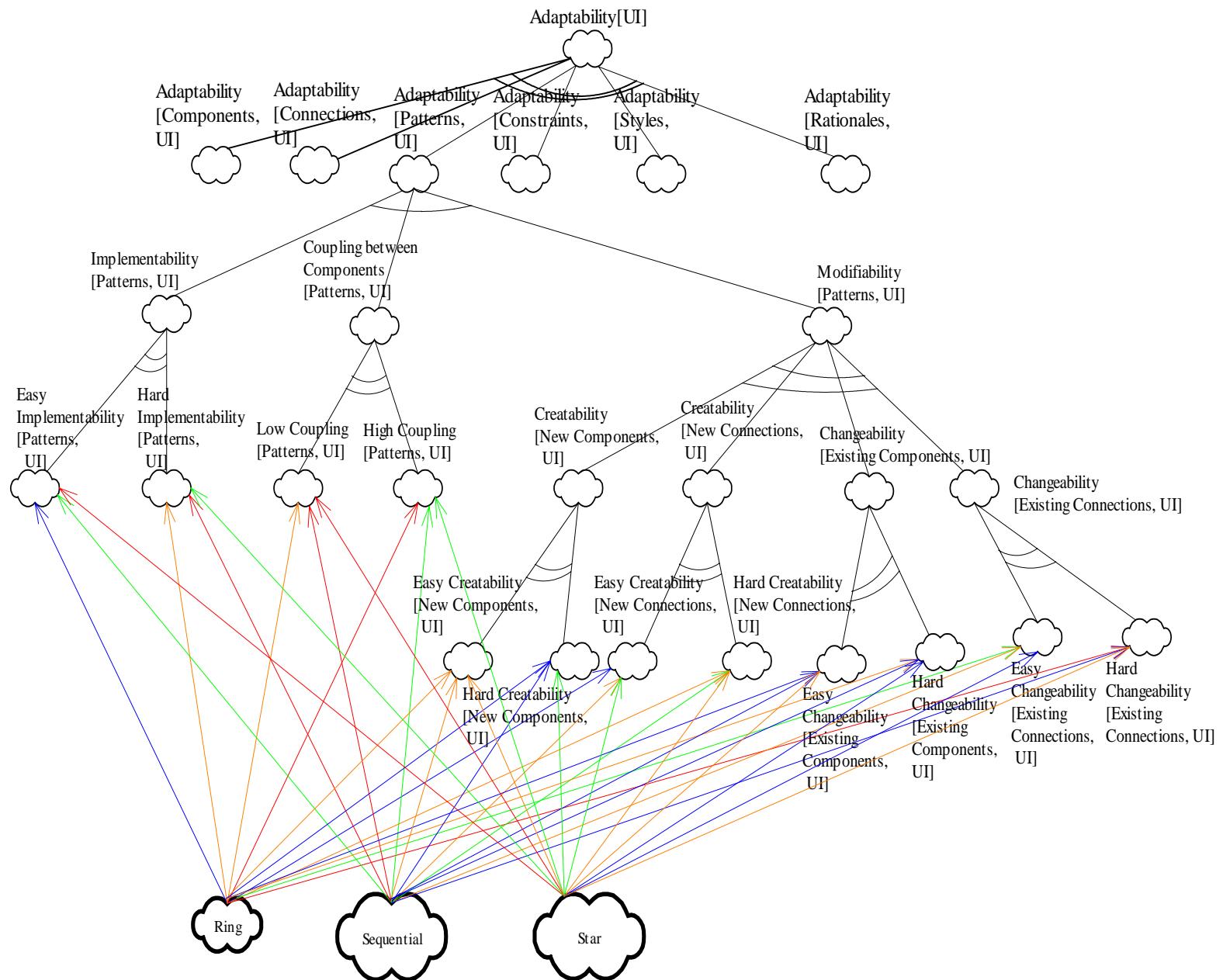


Figure 14. SIG For Patterns

6.4 Populating the KB with Constraints

Figure 15 shows the SIG for batch and sequential constraints.

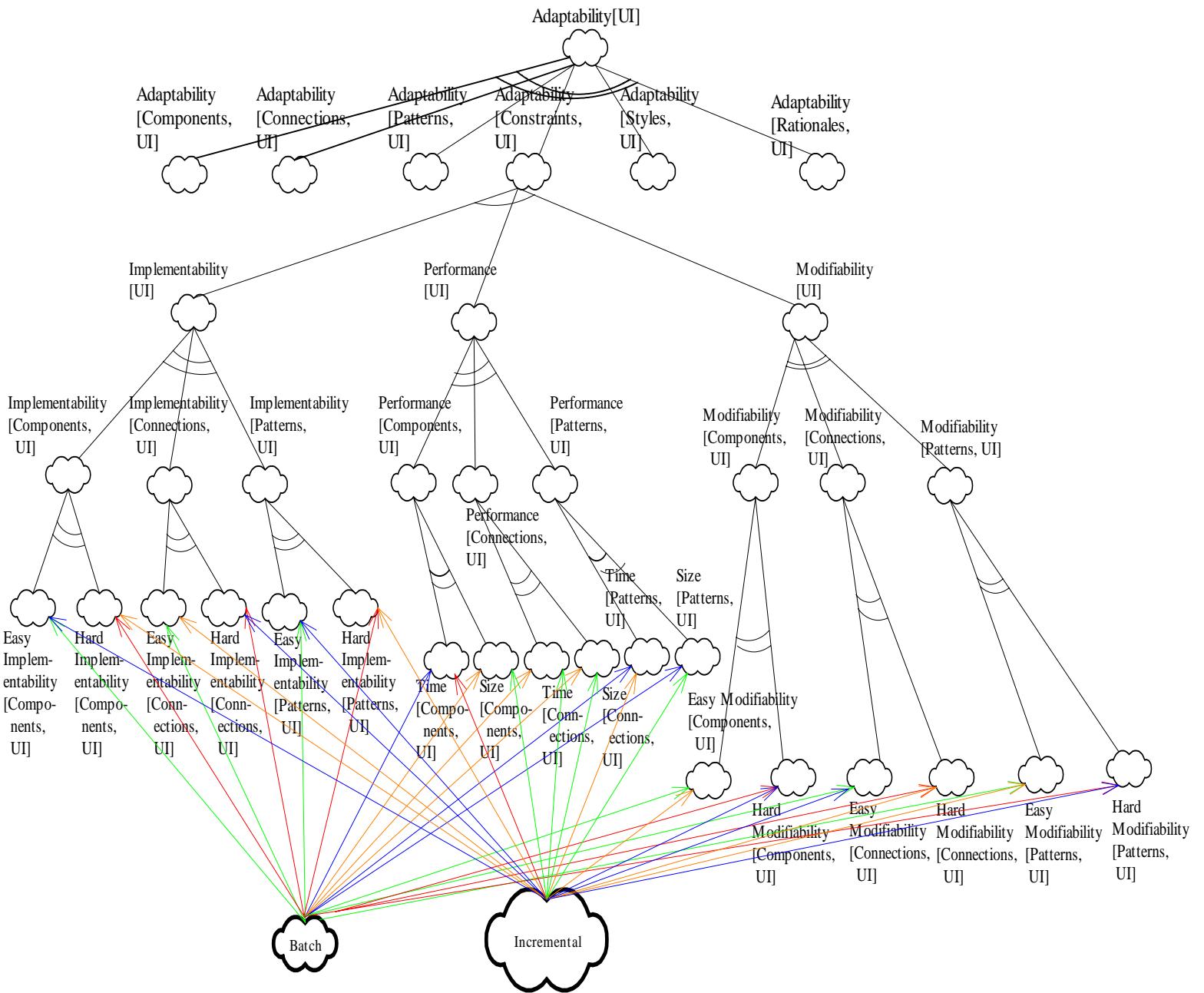


Figure 15. SIG For Constraints

6.5 Populating the KB with Styles

Figure 16 shows the SIG for styles: layered and object-oriented.

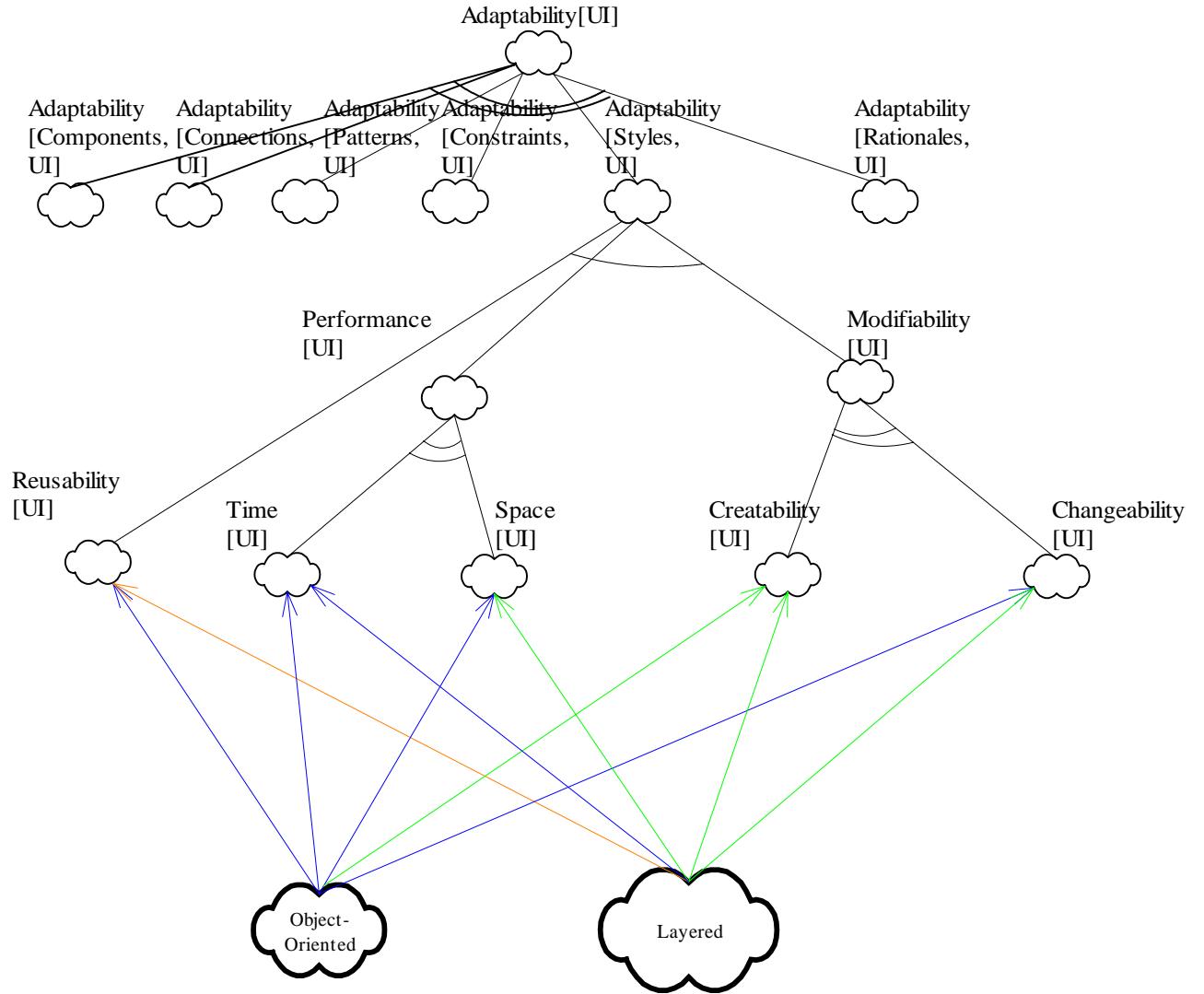


Figure 16. SIG For Styles

6.6 Populating the KB with Rationales

Figure 17 shows the SIG for two rationales: adaptability and performance and how they satisfy adaptability-related NFR softgoals.

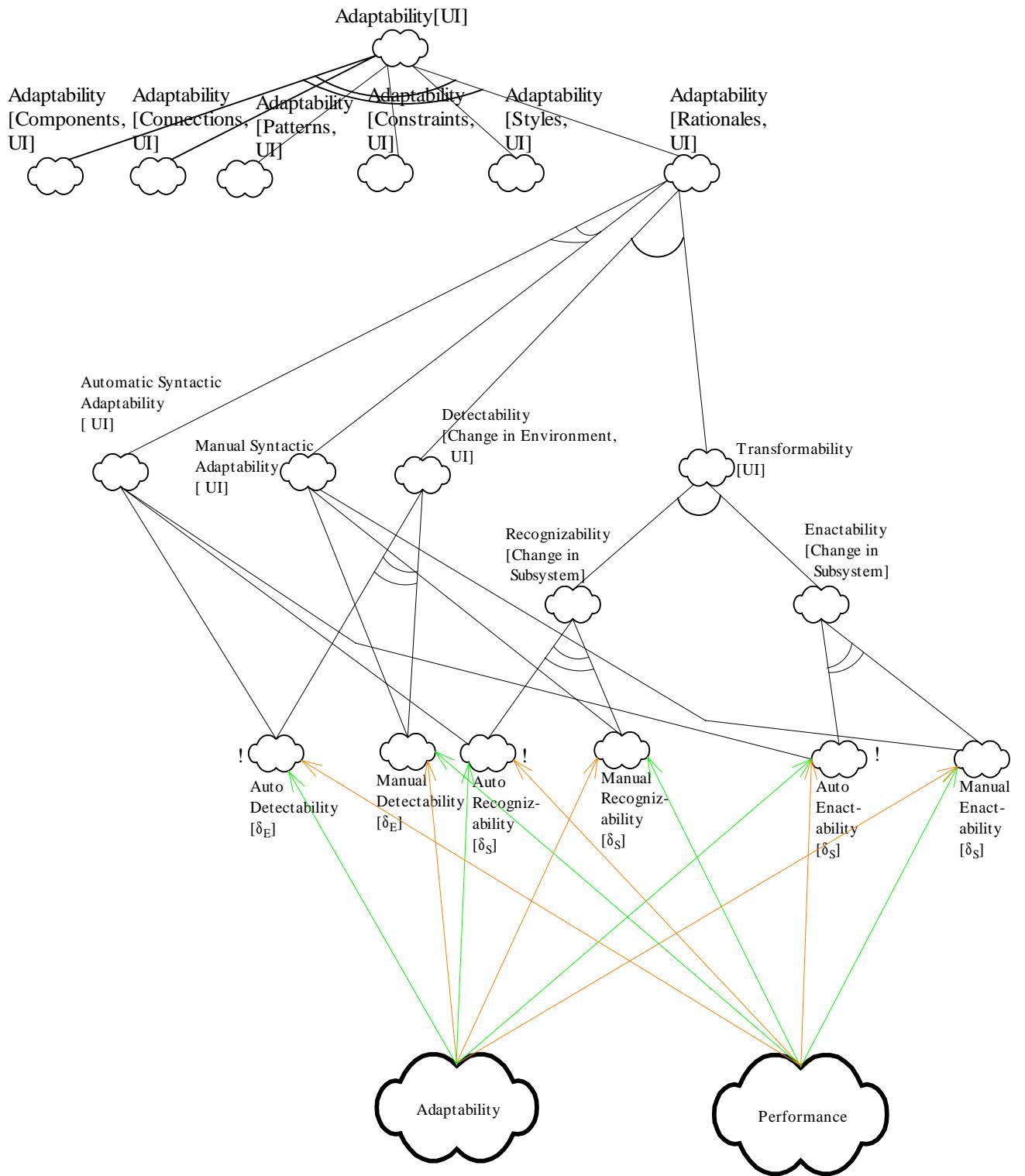


Figure 17. SIG For Rationales

7. Generating Adaptable Architectures for User Interfaces

Now that the KB is populated with catalogs of architectural constituents, the tool SA3 is now ready for use. SA3 generates suitable architectural constituent based on the input NFR from which to begin search. The outputs from SA3 use the convention shown in Figure zz.

Usually the more general the starting NFR softgoal is the more the number of constituents generated by SA3 as the search criteria satisfies a larger population of the KB; the more specific the starting NFR softgoal, the fewer the number of constituents that satifice that softgoal.

There are three levels of searches:

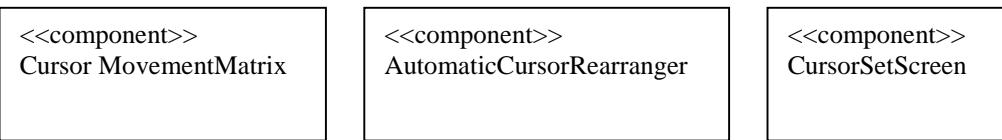
1. first seach for a type of architectural constituent: components, connections, patterns, styles, constraints and rationales
2. then for the chosen architectural constituent search for the extent of satisficing (contribution type): the following choices are available – MAKE, HELP, HURT or BREAK.
3. then a correlation with another NFR softgoal can be given, optionally, as well as the correlation type: the correlations can again be one of MAKE, HELP, HURT or BREAK.

In this section we will demonstrate the automatic adaptable architecture generation capability of SA3 – the architectures generated by SA3 can then be used as a guide and completed manually by the developer. As mentioned earlier, all architectures belong to the UI for test instruments.

7.1 Generating Architectures for User Interfaces with Dynamic Cursor Rearrangement

We wish to generate an architecture that will permit dynamic cursor rearrangement. For the components we started the search with the NFR softgoal

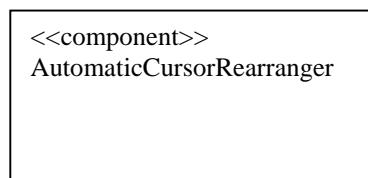
AutoChangeability[SysChange,Rearrangeability[CursorMovement,UpDownKeys,Keys,UI]] and we need components that have MAKE-contribution with this NFR softgoal. The output of the SA3 tool were the following components:



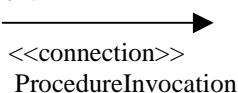
We wanted one more level of resolution: the cursor rearrangement should be detected automatically as well. So we wanted to see if there were components that had a MAKE-contribution with the following NFR softgoal:

AutoRecognizability[EnvChangeRecognition, Rearrangeability[CursorMovement, UpDownKeys, Keys, UI]]

And the result was the following component only:



For the connection, we started with the NFR softgoal EasyImplementability[Connections, UI] and the result was the following MAKE-connection:



For pattern, we looked for patterns that MAKE-satisfied EasyCreatability[New Connections, UI] and the result was

```
<<pattern>>
Star
```

For constraints, we looked for constraints that MAKE-satisfied EasyImplementability[Components, UI] and the result was

```
<<constraint>>
Batch
```

For styles, we looked for MAKE-satisficing of EasyCreatability[UI] and the result was

```
<<style>>
ObjectOriented
```

For rationales, we looked for MAKE-satisficing of AutoDetectability[DeltaE] and the result was

```
?
<<rationale>>
Adaptability
```

With these outputs the developer can create the architecture shown in Figure 18.

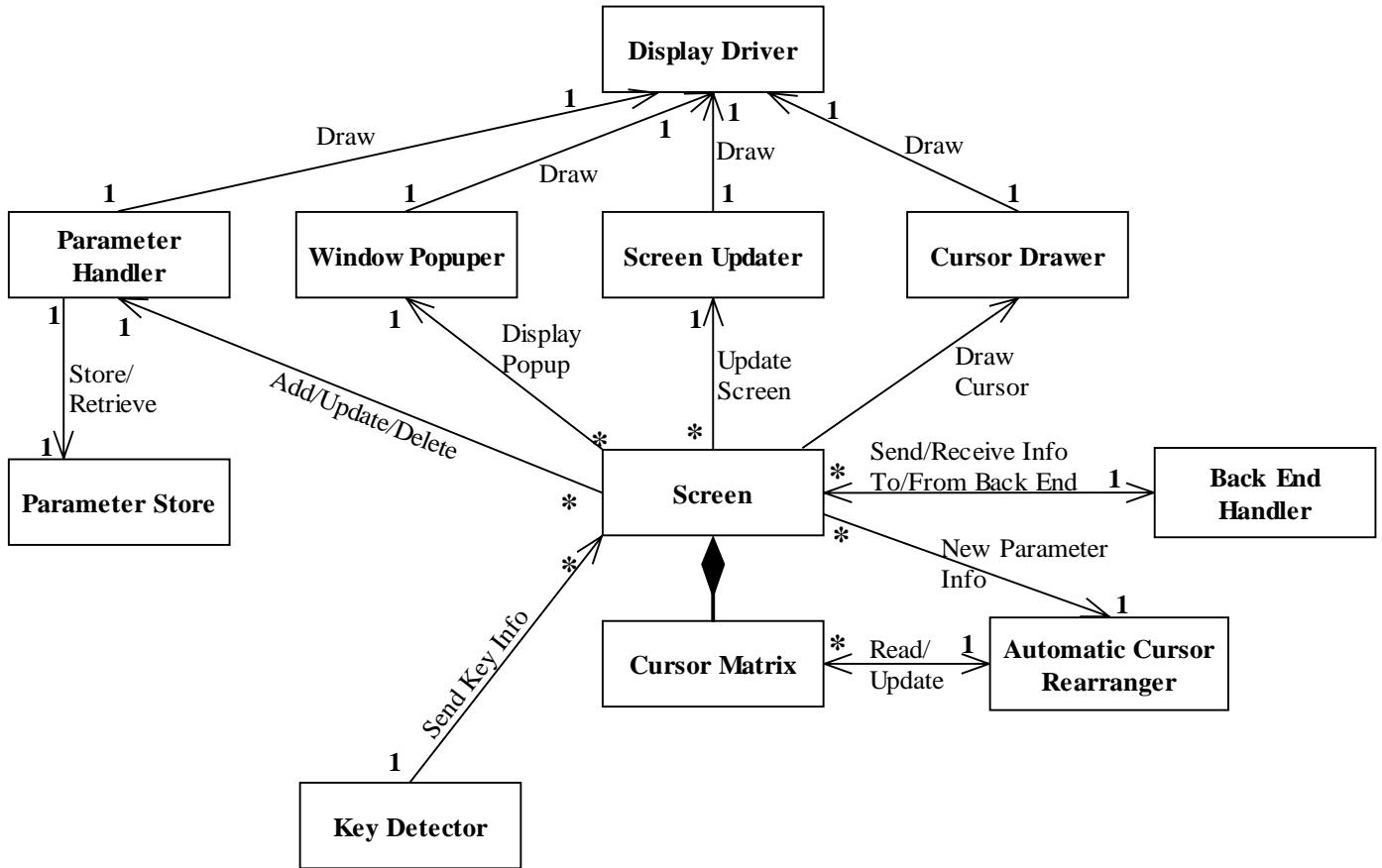


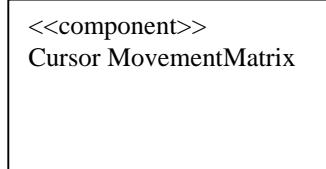
Figure 18. Generated Architecture for Dynamic Cursor Rearrangement

7.1.1 Considering Correlations

In the above example we did not consider correlations. Suppose the correlations for the components were as per the SIG of Figure 19, then when we choose the component we give additional information saying that the component should MAKE-satisfice space requirement, namely, the NFR softgoal Space[Components, UI], then the reply from the SA3 tool would be:

No Components With MAKE-Contribution Could Be Found

The above reply will be received as long as the contribution of the component is HELP or HURT for the NFR softgoal AutoRecognizability[EnvChangeRecognition, Rearrangeability[CursorMovement, UpDownKeys, Keys, UI]]. However, for BREAK contribution for this NFR then we get the following component that also MAKE-correlates with the space requirement:



If this component is used the architecture becomes different and is given in Figure 20.

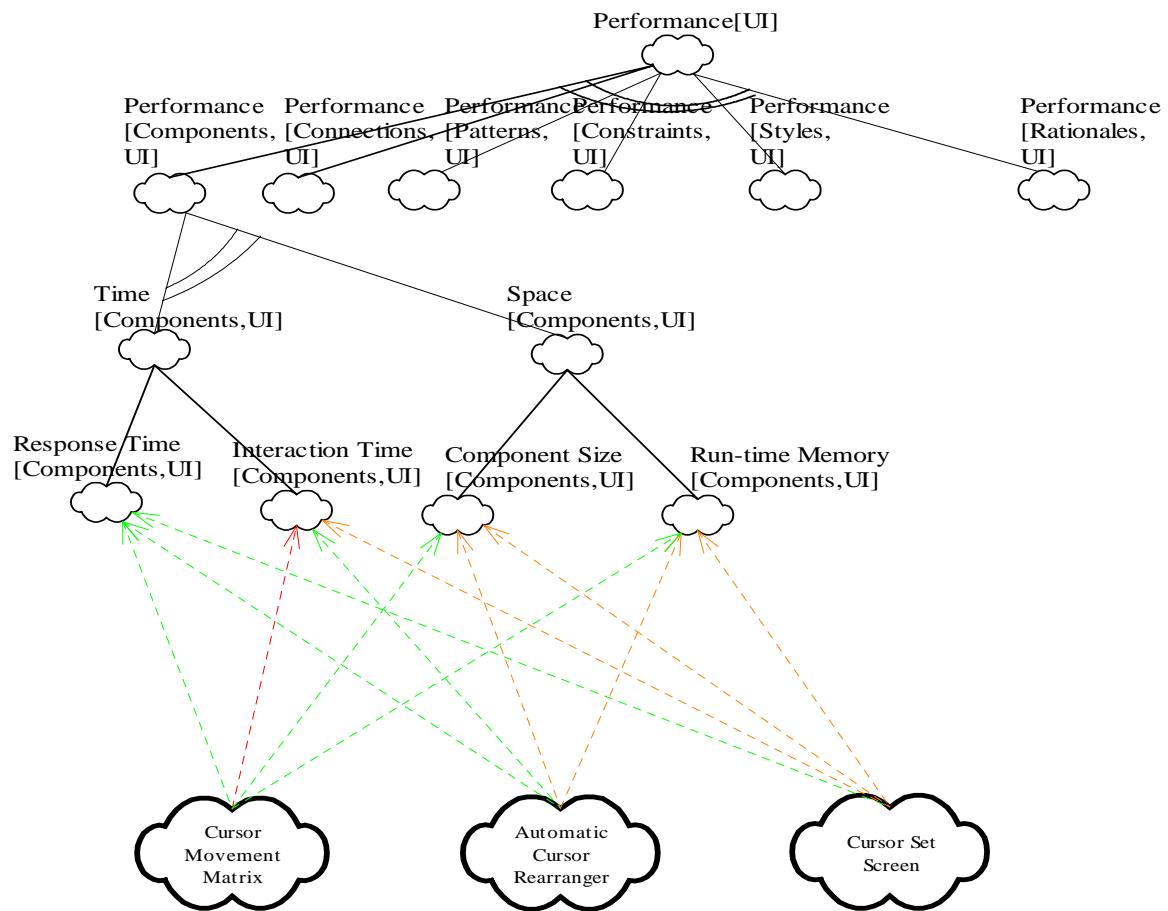


Figure 19. SIG for Correlations

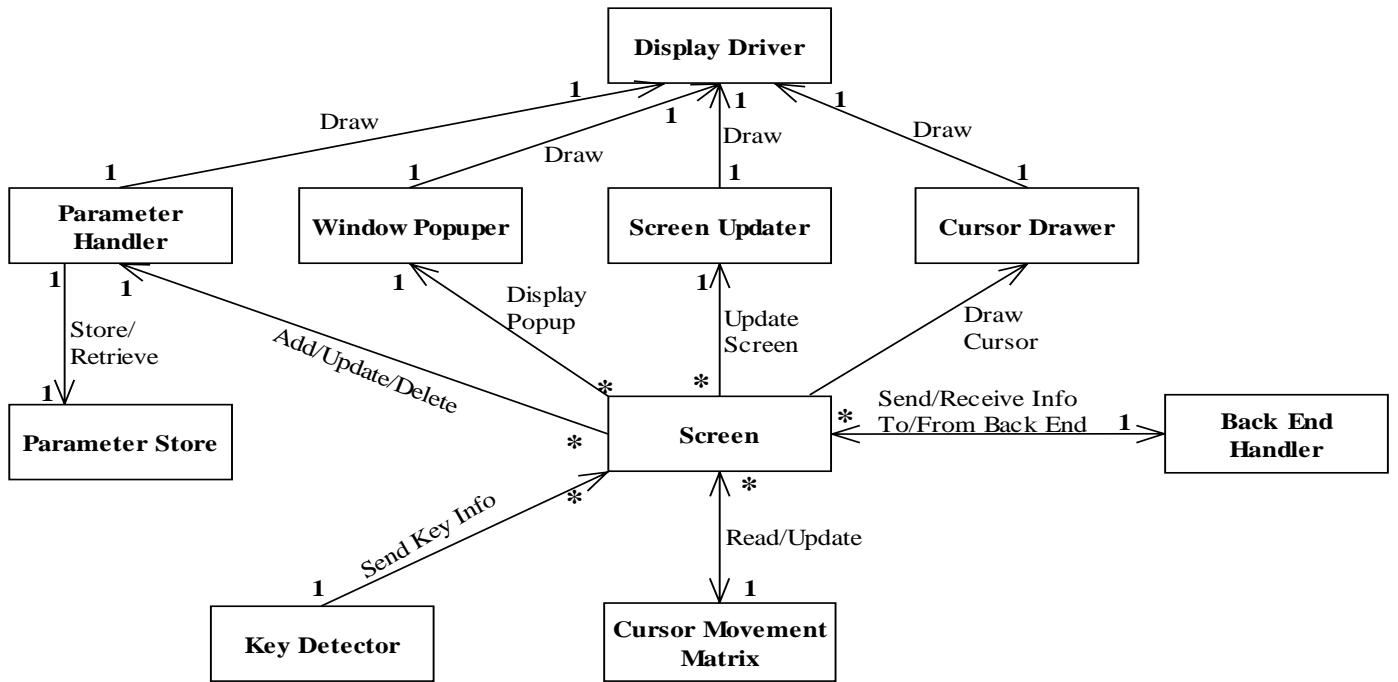


Figure 20. Generated Architecture for Dynamic Cursor Rearrangement with Correlation

7.2 Generating Architectures for Dynamic Field-Width Change

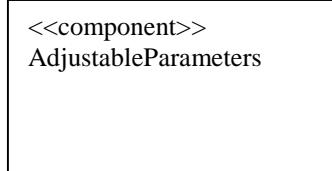
We would like an architecture that automatically changes the field width of parameters or calculated values depending on their lengths. Thus if a parameter x has value 45dBm and not -100dBm, the display will be [45dBm] instead of [-45dBm]. This is referred to as dynamic changing of field-widths. Using the Browsing menu on the menu-bar we can see a list of all NFR softgoals in the KB. For the components, if we choose as the starting NFR softgoal:

AutoRecognizability[EnvChangeRecognition, Modifiability[FieldWidth, Parameters, Display, UI]],

then the following MAKE-components are displayed:



If now the MAKE-correlation for Space[Components, UI] is considered, then the following component is selected:



The other constituents of the architecture were same as before. The architecture of Figure 21 can then be developed.

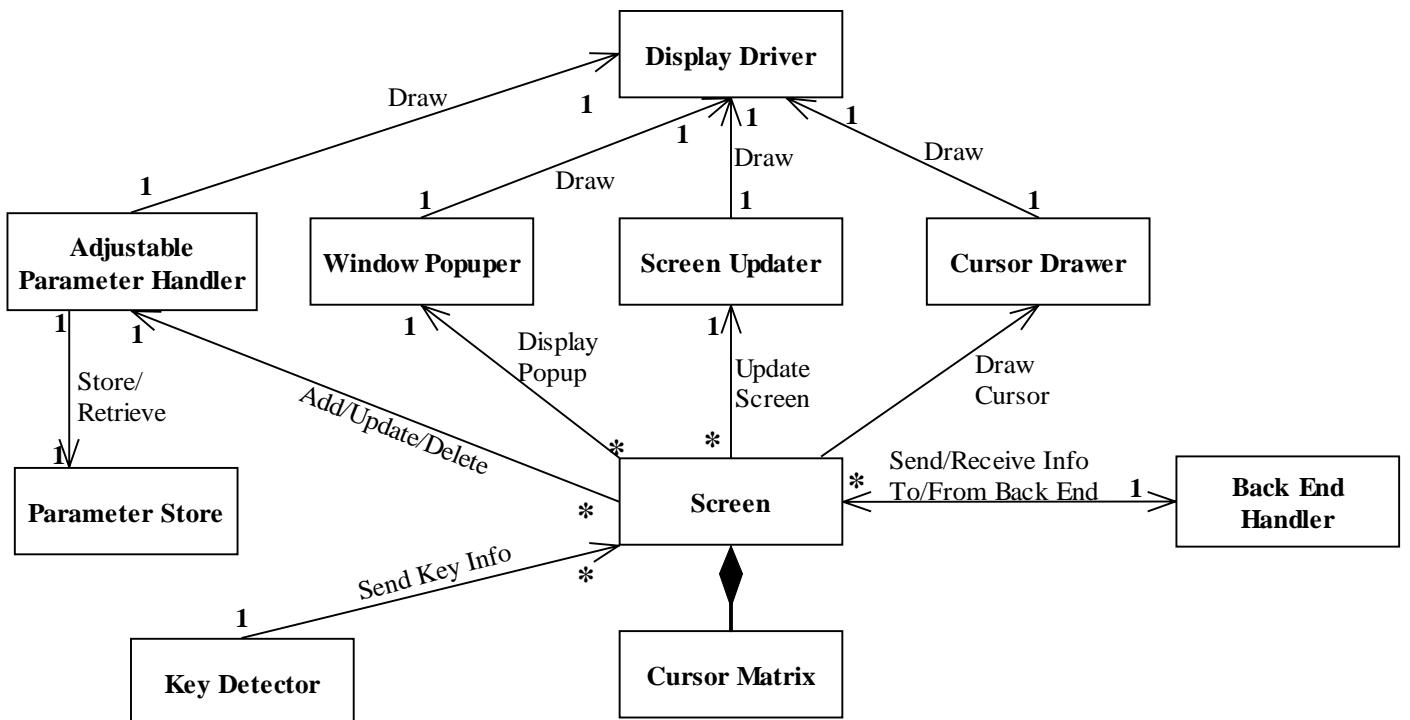
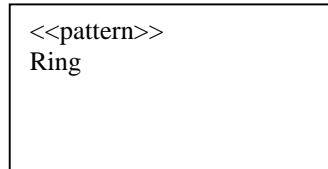


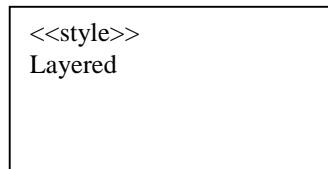
Figure 21. Generated Architecture for Dynamic Field-Width Adjuster

7.2.1 Another architecture for Dynamic Field Width

It is not necessary that the architecture of Figure 21 is the only possible architecture for this feature. Architecture of Figure 21 satisfied certain non-functional requirements with respect to its various constituents. Suppose we varied some of the constituents like the pattern and style what will be the effect be on the architecture? Thus if the pattern had to MAKE-satisfice the NFR softgoal EasyChangeability[Existing Connections, UI] then the output of the SA3 was:



If the style had to MAKE-satisfice the NFR softgoal Space[UI] then the output of the SA3 was:



Then the developer can come up with the architecture of Figure 22, assuming the other constituents were selected as in Section 7.2. Thus the architecture need not be object-oriented and procedural architecture would suffice.

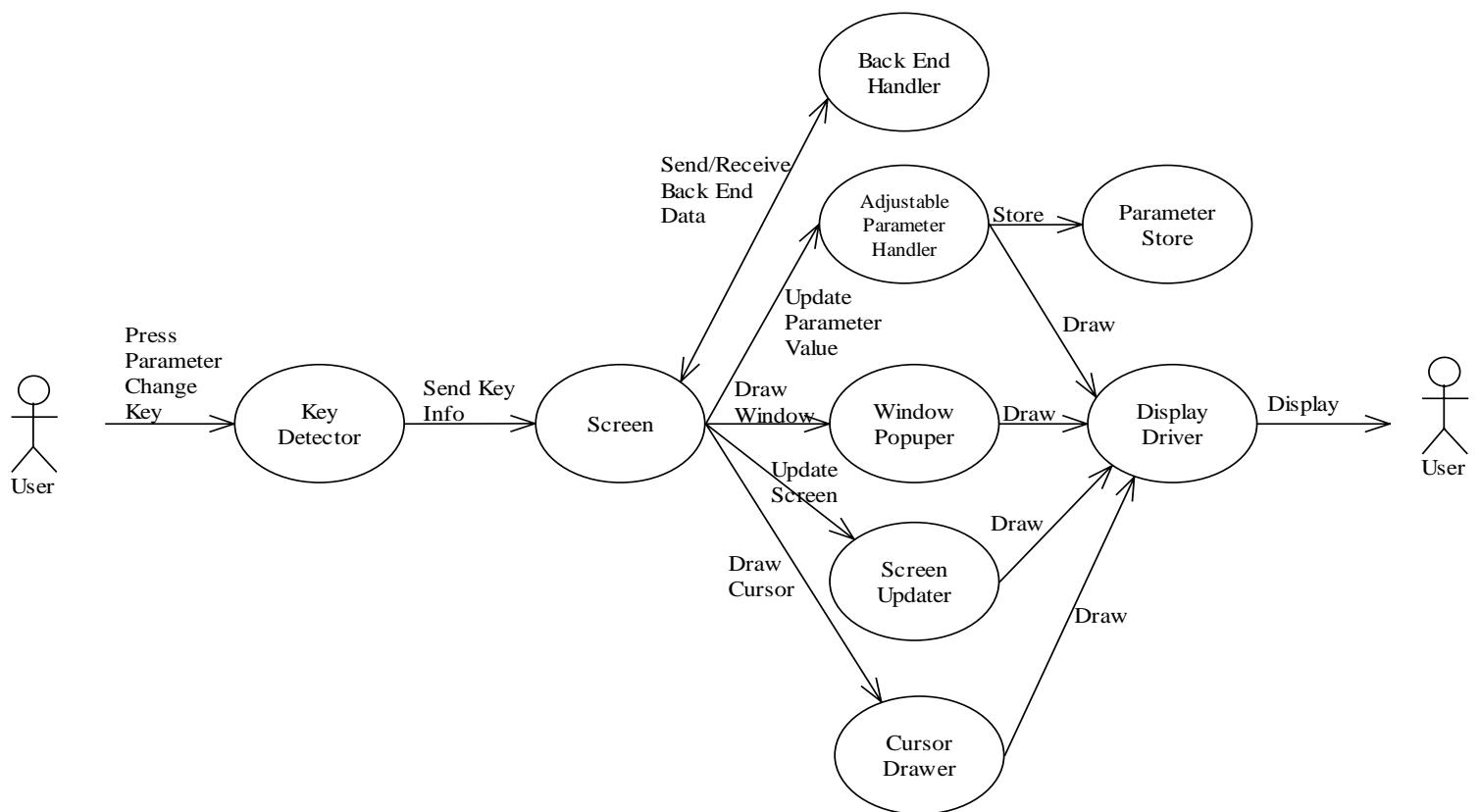
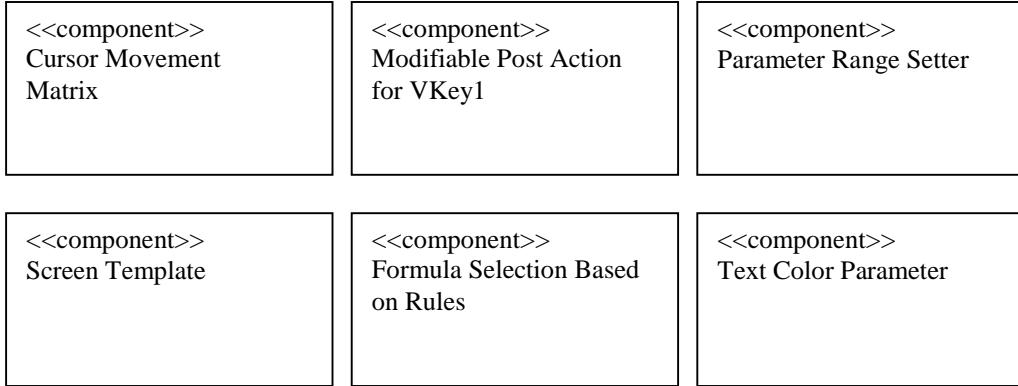


Figure 22. DFD for Dynamic Field-Width Adjustment Feature

7.3 Generating Architecture for Adaptable Screens

In this section we generate an architecture for adaptable screens. We select an NFR softgoal little higher up in the SIG to get a greater selection of architectural constituents. We selected the MAKE-satisficing of the NFR softgoal Adaptability[UI] and we got the following components (we got a lot more – but we are showing only those in the SIG of Figure 13):



Keeping the other architectural constituents as in Section 7.1, we get the architecture for adaptable screens as in Figure 23.

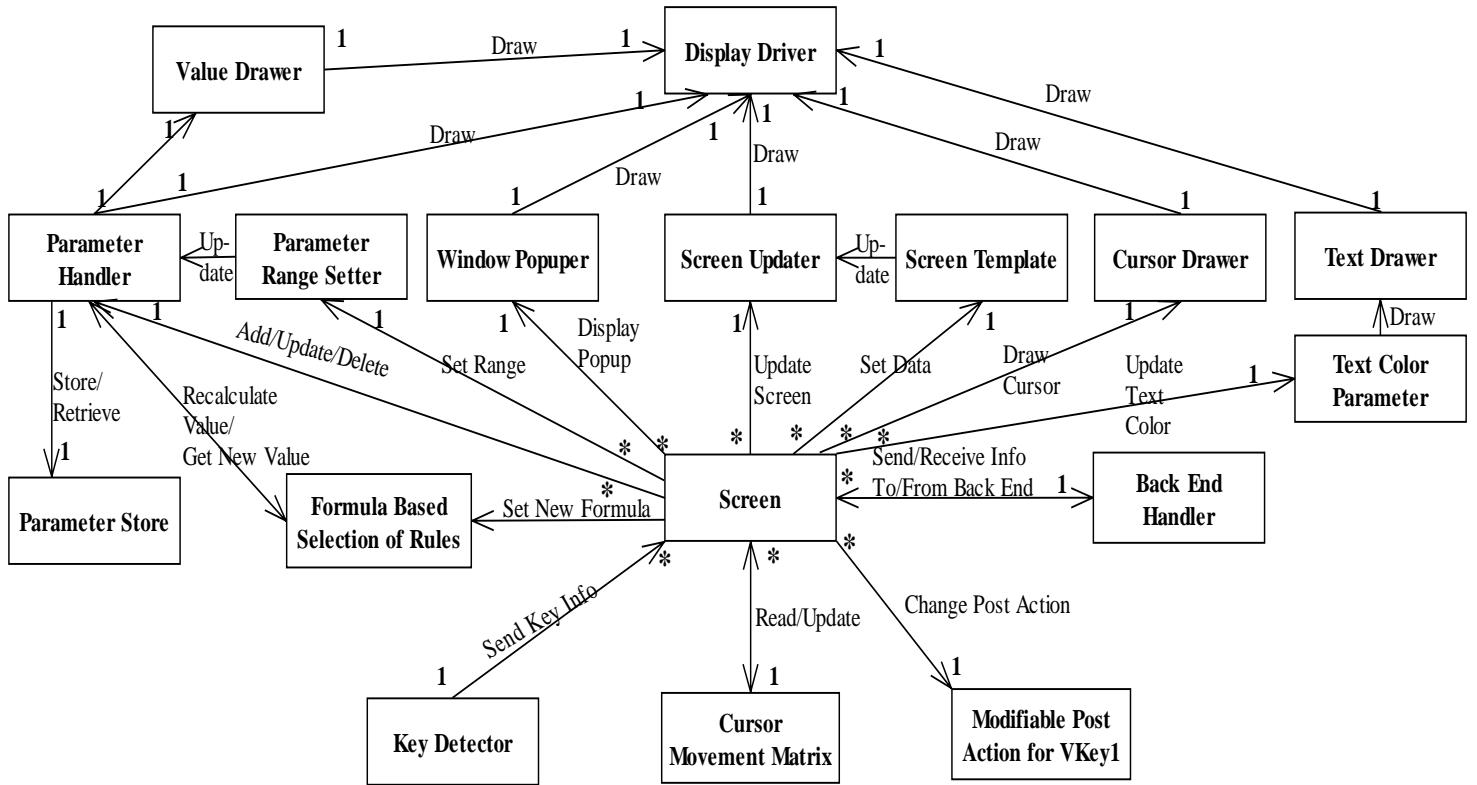


Figure 23. Architecture for Adaptable Screens

8. Validation of Generated Architectures

In this section we validate the architectures generated in Section 7 by implementing them in a test instrument and checking to ensure that the adaptability exists.

8.1 Validating Dynamic Cursor Movement Rearrangement

The architecture of Figure 10 was implemented and vertical key VKey1 was attached to a popup menu that lets one add new parameters. Upon pressing the soft key “Add New Parameter” the popup menu as in Figure 24 popups up. When data is entered (as shown in Figure 24) the resulting screen is shown in Figure 25. In the screen of Figure 25 the cursor movement has been dynamically changed. There is no need to explicitly enter the new cursor movement matrix.

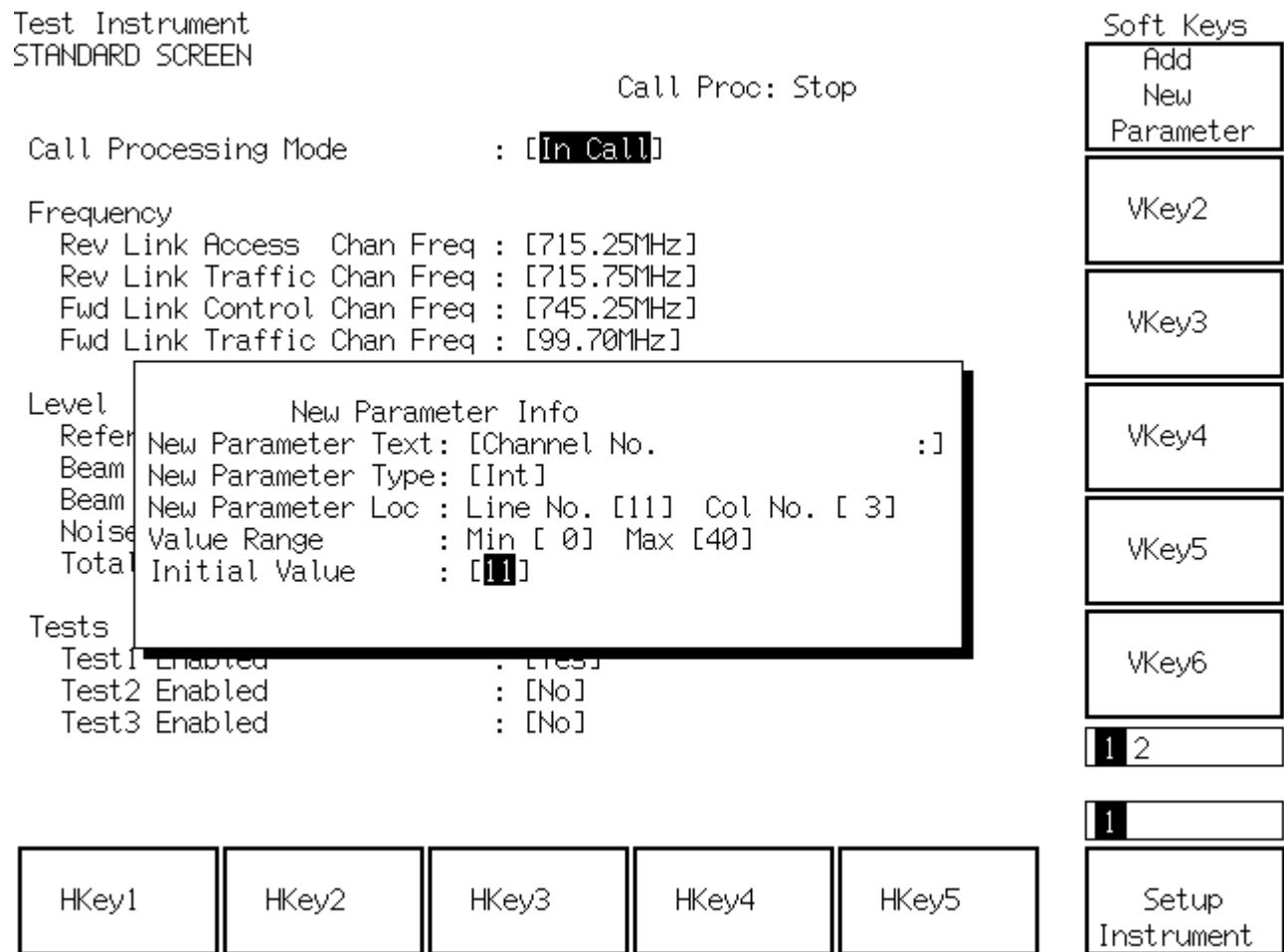


Figure 24. Popup For Adding New Parameter

Test Instrument
STANDARD SCREEN

Call Proc: Stop

Call Processing Mode : [In Call]

Frequency

Rev Link Access Chan Freq : [715.25MHz]
Rev Link Traffic Chan Freq : [715.75MHz]
Fwd Link Control Chan Freq : [745.25MHz]
Fwd Link Traffic Chan Freq : [99.70MHz]
Channel No. : [11]

Level

Reference Level : [-5.0dBm]
Beam 1 Level (Ior1) : [-8.5dBm]
Beam 2 Level (Ior2) : [-6.0dBm]
Noise Level (Ioc/1.23 MHz) : [-9.0dB]
Total Output Level : (-25.7dBm)

Tests

Test1 Enabled : [Yes]
Test2 Enabled : [No]
Test3 Enabled : [No]

Soft Keys

Add
New
Parameter

VKey2

VKey3

VKey4

VKey5

VKey6

1 2

1

HKey1

HKey2

HKey3

HKey4

HKey5

Setup
Instrument

Figure 25. Screen with new Parameter Added

8.1.1 Manually Changing the Cursor Movement Matrix

The architecture of Figure 20 was implemented and vertical key VKey2 was attached to a popup screen that showed the new cursor arrangement. When VKey2 was pressed the screen as in Figure 26 popped up. When the new cursor movement matrix was entered in this popup the subsequent key movement obeyed the new cursor movement matrix.

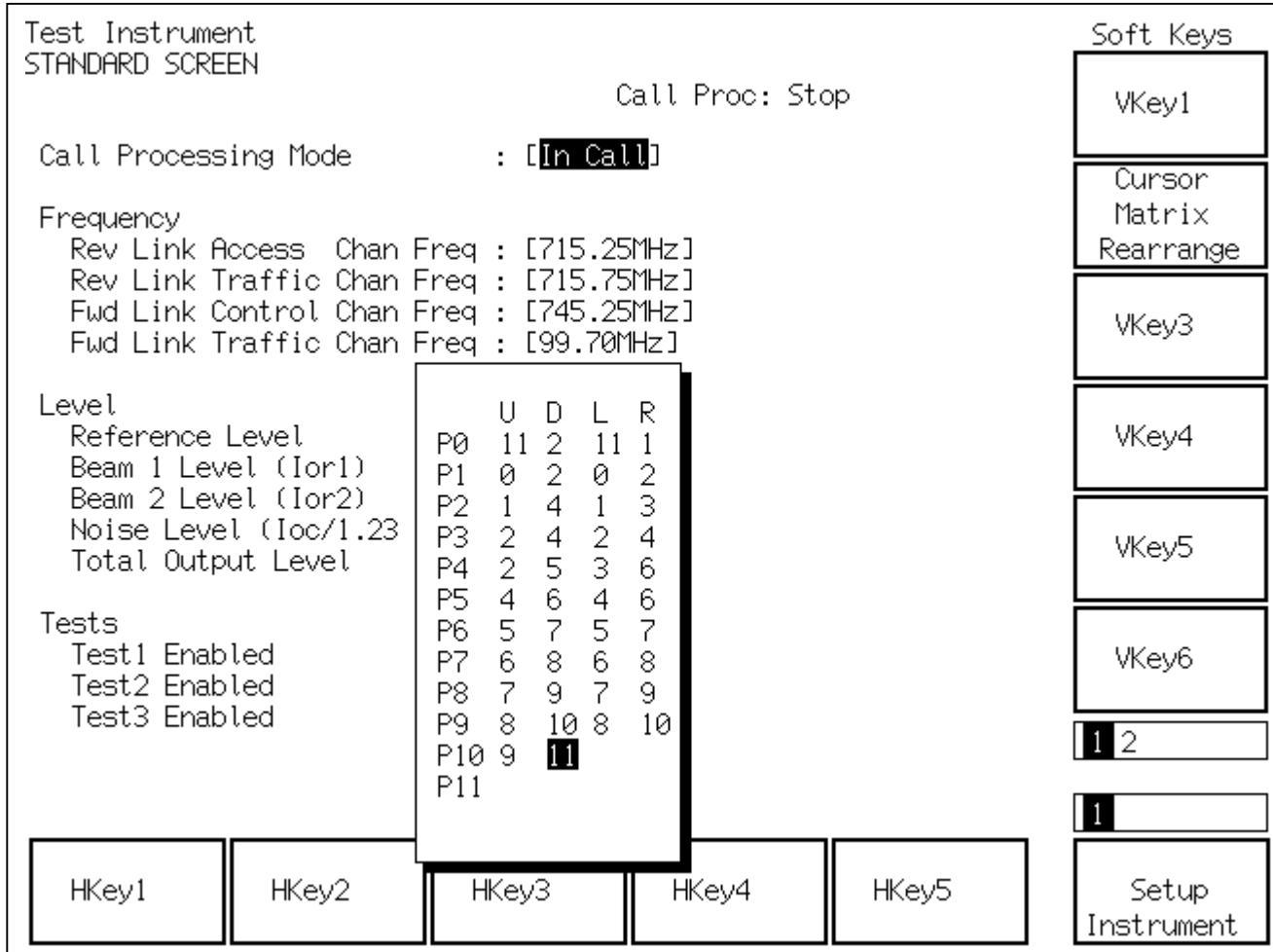


Figure 26. Screen with Popup for Cursor Movement Rearrangement

In Figure 2 the cursor moved from first parameter (Call Processing Mode) to the last parameter in a sequence in response to up, down, left and right keys (for this screen, the left key implemented the same movement as the up key, while the right key implemented the same movement as the down key). However, due to a study of use-patterns by the customers, it was decided that the parameters Rev Link Traffic Chan Freq and Fwd Link Traffic Chan Freq are used more frequently than the others. So in order to implement this new cursor movement, a vertical soft key called "Cursor Matrix Rearrange" was created – upon pressing of this key a popup as in Figure 26 is displayed that lets the user enter the new cursor movement matrix (on the left on this popup are the parameters indexed from 0 and across the top are the Up, Down, Left and Right keys). The matrix indicates the parameter jumps for the corresponding cursor key presses.

The advantage of the architecture of Figure 20 is in the lower number of bytes taken up by the binary executable. A savings in the order of about 10000 bytes (out of the total system size of about 100K bytes) was possible.

8.2 Validating Dynamic Field Width Feature

8.2.1 Before Adaptation

Figure 27 shows the field widths before adaptation for this feature.

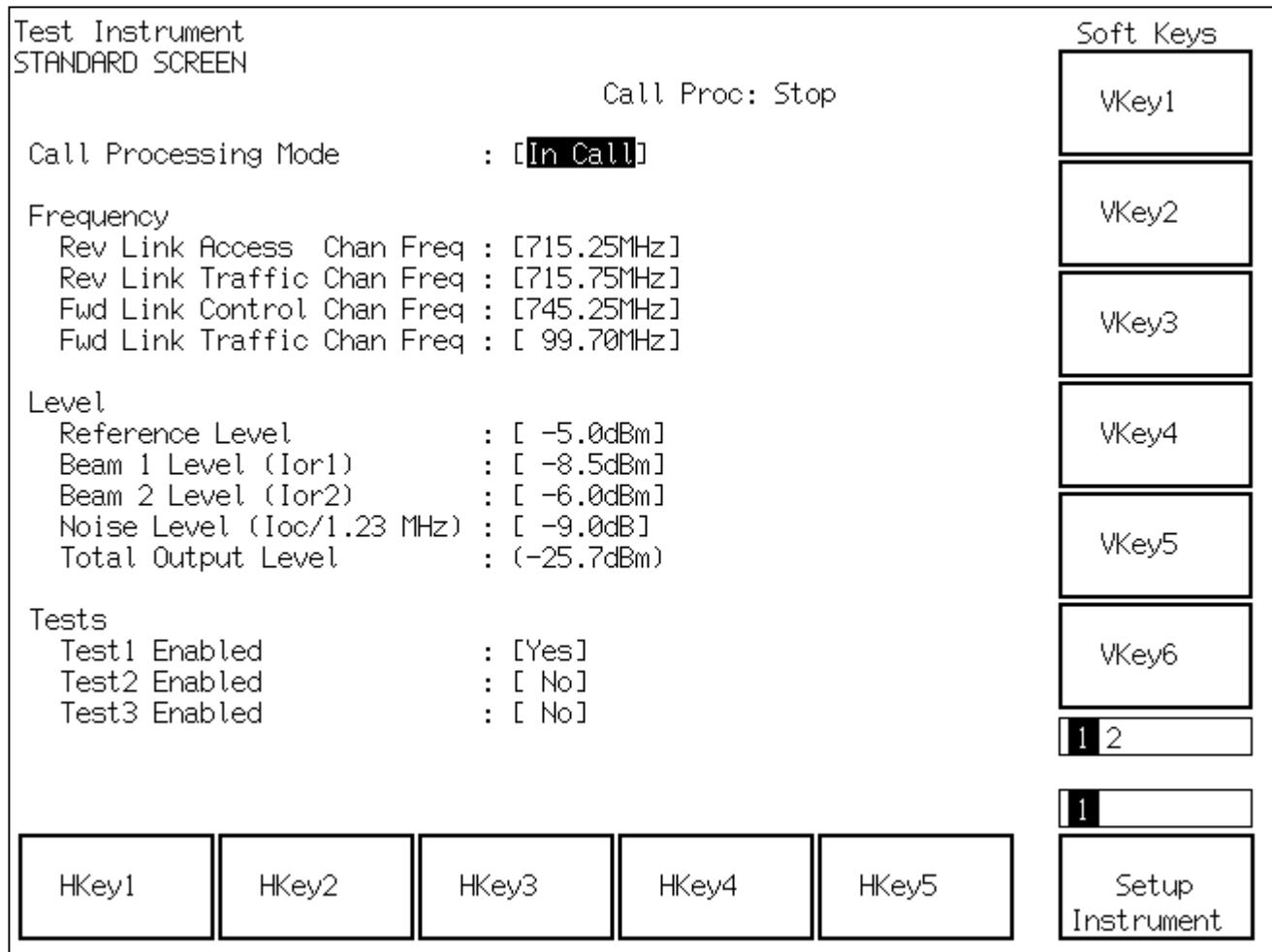


Figure 27. Screen Display before Adaptation for Dynamic Field Width

As can be seen in Figure 27, even though the values for the parameters Fwd Link Traffic Chan Freq, Reference Level, Beam 1 Level, Beam 2 Level and Nose Level have fewer digits than their values in Figure 1, the field width of the parameters remains the same as before. This may or may not be desirable in some cases.

8.2.2 After Adaptation

Using the architecture of Figure 21, the screen looks like Figure 27. As can be seen the parameters field width has adapted to their new sizes.

As can be seen from Figure 28 the widths of the affected parameters including those of Test2 Enabled and Test3 Enabled have changed.

Test Instrument STANDARD SCREEN		Call Proc: Stop	Soft Keys		
Call Processing Mode		: [In Call]	VKey1		
Frequency			VKey2		
Rev Link Access Chan Freq		: [715.25MHz]	VKey3		
Rev Link Traffic Chan Freq		: [715.75MHz]			
Fwd Link Control Chan Freq		: [745.25MHz]			
Fwd Link Traffic Chan Freq		: [99.70MHz]			
Level					
Reference Level		: [-5.0dBm]	VKey4		
Beam 1 Level (Ior1)		: [-8.5dBm]			
Beam 2 Level (Ior2)		: [-6.0dBm]			
Noise Level (Ioc/1.23 MHz)		: [-9.0dB]			
Total Output Level		: (-25.7dBm)			
Tests			VKey5		
Test1 Enabled		: [Yes]			
Test2 Enabled		: [No]	VKey6		
Test3 Enabled		: [No]			
			1 2		
			1		
HKey1	HKey2	HKey3	HKey4	HKey5	Setup Instrument

Figure 28. Screen Display after Adaptation for Dynamic Field Width

8.3 Validating Adaptable Screens

In this section we will validate some of the adaptable characteristics of the architecture of Figure 23.

8.3.1 Validating Formula Selection

Vertical softkey VKey3 is tied to “Select New Formula”. When this key is pressed a pop-up box appears listing the various formula applicable for the current screen. Any calculated parameter on the screen has an allowable set of formulas applicable to it. Thus from the pop-up screen when a formula is selected it will apply to that calculated parameter to which the formula is applicable. Figure 29 shows the screen with a pop-up that helps selecting from among formulas applicable for the calculated parameter “Total Output Level”.

8.3.2 Validating Screen Template

Vertical softkey VKey4 is tied to “Screen Template” (Figure 29). When this key is pressed it displays a new screen template with the title “New Screen Template”. This template could be used to create a new screen or to change the appearance of current screens. Figure 30 shows the New Screen Template.

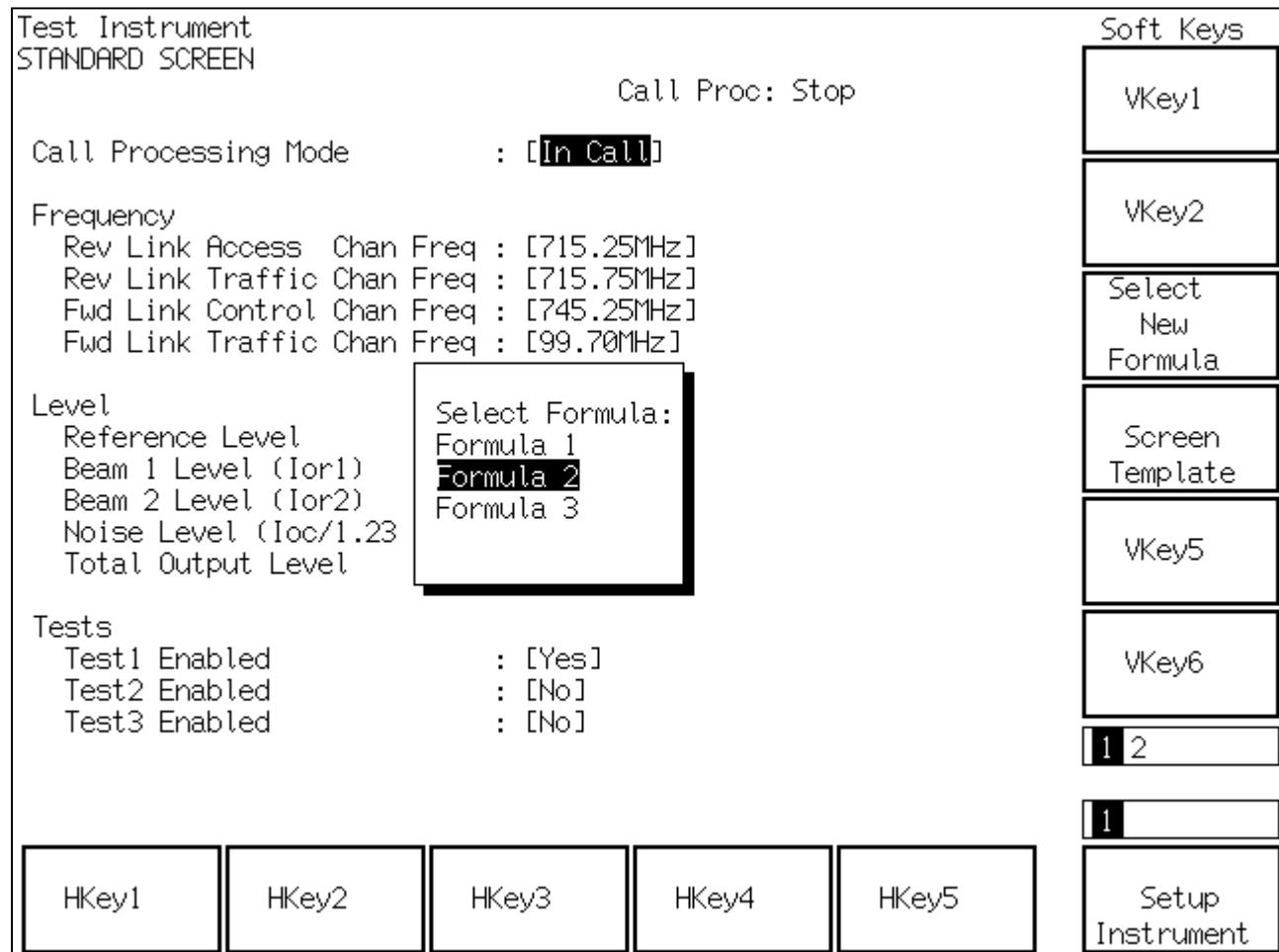


Figure 29. Popup for Formula Selection

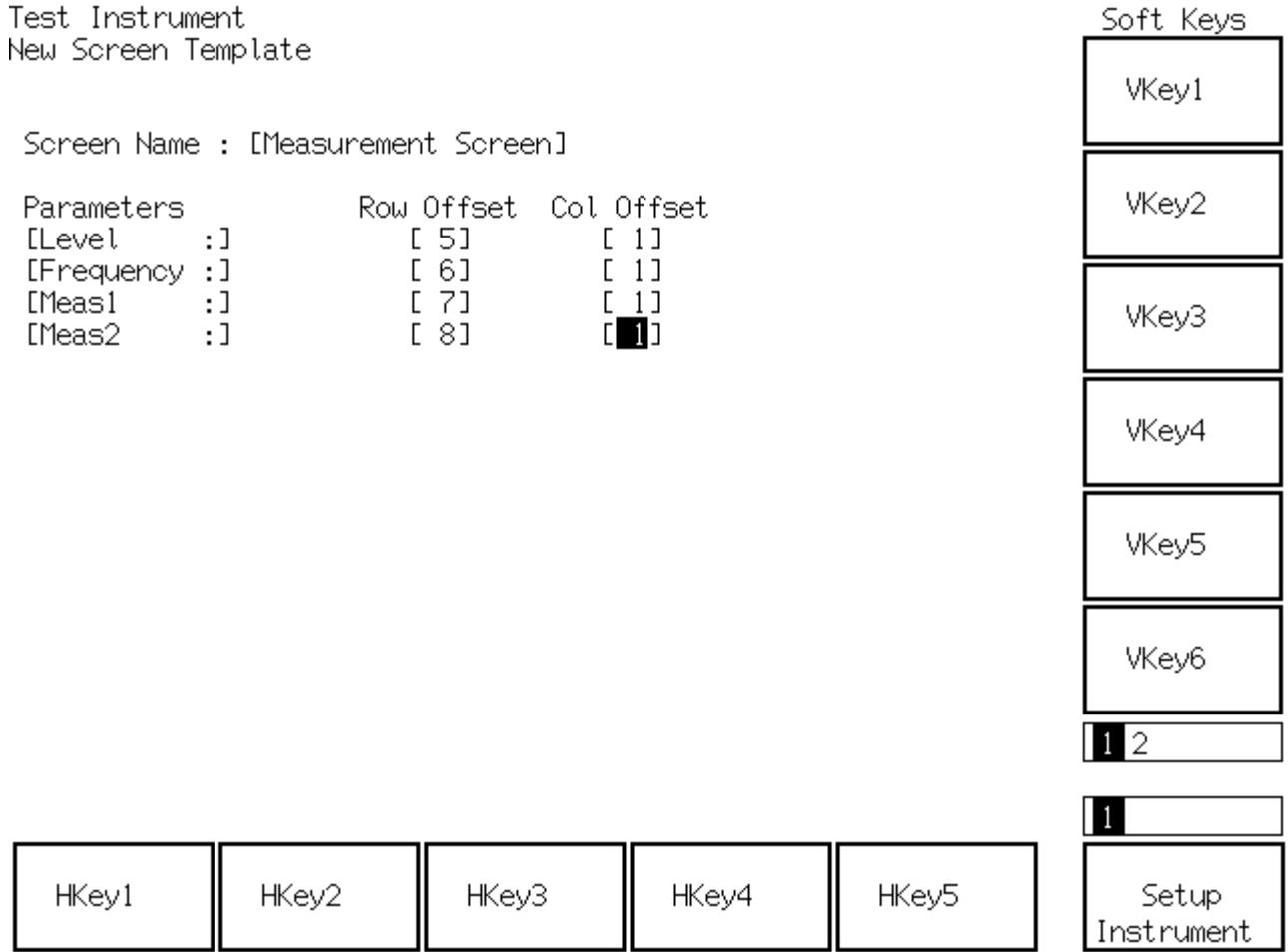


Figure 30. Template for Adding New Screen

8.4 Observations

We found that the tool is useful in generating adaptable architectures – the tool is also a perfect example of reuse of design [28,29]; however, here the reuse is guided by non-functional requirement of adaptability instead of the functional requirements, as is usually the case. By reusing designs we also ensure that the knowledge gained by the software developing organization is not wasted but put to good use.

The time to develop the tool is not a major consideration in this approach. Once the tool is developed, the tool can be used as is for as long as the developer/her organization wants. We took about 2 months' work to develop the tool; but we have been using this tool for the past 6 months on different domains.

One of the apparent negative point in the use of the tool is the time taken to populate the knowledge base. For some of the SIGs, it took us about 2-3 hours of work to populate its elements into the knowledge base. However, once populated that knowledge was stored for ever; and that knowledge was also reusable for ever. This was true even when the tool was modified to add extra features.

One question that may come to users mind is how to find the NFR softgoals to start a search. This is where the browsing feature of SA3 comes in handy. The browser helps display all the instances of NFR softgoals in the system. From this list one can determine the NFR softgoal to start searches from.

Another feature of the tool is that the more general the softgoal used to start the search from the more wider the choices presented. Thus in order to restrict variety the user may have to choose a more specific adaptability-related NFR softgoal that the design should meet (or more accurately, satisfice).

Another point to note is that more the population of the knowledge base the better the architecture developed by SA3 reflects reality. This is because the searches come up with more and more relevant architectural constituents. Also with the addition of relevant correlation rules in the knowledge base, it will be easy to identify which the trade-off's made in order to achieve adaptability.

And finally it should be noted that the architectural constituents the searches come up with all satisfy NFRs related to adaptability. Hence the architecture developed using these searched constituents will be adaptable as well.

9. Conclusion

Recently adaptability has emerged as an important quality attribute that almost all software systems are required to possess, particularly, embedded systems. Briefly, adaptability is the ability of a software system to accommodate changes in its environment. Embedded systems are usually constrained in both hardware and software as a result of which developing adaptable architectures for them presents challenges of its own. One of the reasons for this is the lack of comprehensive systematic methods to develop adaptable architectures. The NFR Framework [9,10,11] provides a method that helps to systematically consider NFRs such as adaptability during the process of architecture development. Moreover, the knowledge base properties of the NFR Framework lend themselves to automation. In this paper we exploit the knowledge base properties of the NFR Framework by developing a tool called the Software Architecture Adaptability Assistant (SA3) that helps to automate the generation of adaptable architectures for embedded systems. SA3 helps the developer during the process of software architecture development by choosing the architectural constituents such as components, connections, patterns, constraints, styles, and rationales [7,8] that best fit the adaptability requirements. The developer can then complete the architectures, if required.

In order to demonstrate the architecture generating ability of SA3 we used the tool in the domain of user interfaces – in particular, user interfaces for embedded systems. We populated the knowledge base with various architectural constituents for user interface architectures and we also populated the knowledge base with the information to what extent these architectural constituents satisficed (in the parlance of the NFR Framework) the various adaptability-related non-functional requirements. We also populated the knowledge related to the correlation of the architectural constituents with other synergistic or conflicting non-functional requirements such as performance, security, etc. With this knowledge, SA3 was able to generate adaptable architectural constituents which the developer could then complete. The final architecture developed is assured to be adaptable since its constituents were themselves adaptable in the first place. The architectures so developed were validated by implementing them in a real embedded system – a test equipment used for testing cell phones. The user interface for this test equipment displayed the adaptability characteristics expected of it before the implementation (the various screen shots of the user interface are shown).

We feel that SA3 is a promising tool to semi-automatically generated adaptable architectures. We would like to test this tool further in different domains. We would like to add more correlation capability to this tool so that more interesting NFR trade-off's may be analyzed while developing adaptable architectures. However, we believe that the tool may be the first step towards realizing the goal of automatically developing adaptable architectures for software systems.

References

1. S. Treu, User Interface Design: A Structured Approach, Plenum Press, New York, 1994, p. 196-197.
2. F. Newberry-Paulish and W.F.Tichy, EDGE: An Extensible Graph Editor, Software-Practice & Experience, 20(6), p. 63-88, July 1990.

3. S. Stille, S. Minocha, and R. Ernst, A²DL – Adaptive Automatic Display Layout System, in Proc. of 3rd Annual IEEE Symposium on Human Interactions with Computer Systems (HICS'96), IEEE Computer Press, Los Alamitos, 1996.
4. E. Kandogan and B. Shneiderman, Elastic Windows: Improved Spatial Layout and Rapid Multiple Window Operations, in Proc. of 3rd Int. ACM Workshop on Advanced Visual Interfaces AVI'96 (Gubbio, 27-29 May 1996), ACM Press, New York, 1996, pp. 29-38.
5. T. Miah and J.L. Alty, Vanishing Windows: An Empirical Study of Adaptive Window Management, Computer-Aided Design of User Interfaces II, Eds. J. Vanderdonckt and A. Puerta, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999, pp. 171-184.
6. P.J. Barclay, T. Griffiths, J. McKirdy, N.W.Paton, R.Cooper, J. Kennedy, The Telleach Tool: Using Models For Flexible User Interface Design, Computer-Aided Design of User Interfaces II, Eds. J. Vanderdonckt and A. Puerta, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999, pp.139-157.
7. Shaw, M., and Garlan, D., *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.
8. Bass, L., Clements, P., and Kazman, R., *Software Architecture in Practice*, SEI Series in Software Engineering, Addison-Wesley, 1998.
9. Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J. *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Boston, 2000.
10. Mylopoulos, J., Chung, L., Liao, S.S.Y., Wang, H., Yu, E. “Exploring Alternatives During Requirements Analysis”, *IEEE Software*, Jan/Feb. 2001, pp. 2 – 6.
11. Mylopoulos, J., Chung, L., Nixon, B. “Representing and Using Nonfunctional Requirements: A Process-Oriented Approach”, *IEEE Transactions on Software Engineering*, Vol. 18, No. 6, June 1992, pp. 483-497.
12. P. Savolainen, H. Konttinen, A Framework for Management of Sophisticated User Interface’s Variants in Design Process, Computer-Aided Design of User Interfaces II, Eds. J. Vanderdonckt and A. Puerta, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999, pp.205-215.
13. E. Lecolinet, XXL: A Visual+Textual Environment for Building Graphical User Interfaces, Computer-Aided Design of User Interfaces II, Eds. J. Vanderdonckt and A. Puerta, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999, pp.115-126.
14. S. Kovacevic, Beyond Automatic Generation – Exploratory Approach to UI Design, Computer-Aided Design of User Interfaces II, Eds. J. Vanderdonckt and A. Puerta, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999, pp. 79 – 95.
15. G. Patry and P. Girard, “GIPSE, A Model Based System for CAD Software”, Computer-Aided Design of User Interfaces II, Eds. J. Vanderdonckt and A. Puerta, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999, pp. 61 – 72.
16. N. Subramanian and L. Chung, “Tool Support for Engineering Adaptability into Software Architecture”, Proceedings of the International Workshop on Principles of Software Evolution, ACM Press, May 2002, pp.
17. Subramanian, N., and Chung, L. “Software Architecture Adaptability – An NFR Approach”, to appear in the *Proceedings of International Workshop on Principles of Software Evolution*, Vienna, September 2001.
18. Booch, G., Rumbaugh, J., Jacobson, I., 1999. The Unified Modeling Language User Guide, Addison-Wesley, Reading, Massachusetts.
19. www.agilent.com
20. www.anritsu.com
21. Laplante PA. *Real-Time Systems Design and Analysis*. IEEE Press : Piscataway, New Jersey, 1992; 1-18.
22. Application Note No. AN2019/D, MC68302 Design Concept – Expanding Interrupts on the MC68302, July, 1990, available from www.motorola.com
23. A. Downtown (Ed.), Engineering in Human-Computer Interface, McGraw-Hill Book Company, Beshire, England, 1991.
24. E. A. Edmonds, Adaptive Man-Computer Interfaces, in Computing Skills and the User Interface, Edited by M. J. Coombs and J. L. Alty, Academic Press, London, p. 389 – 426.
25. R. C. Thomas, The Design of an Adaptable Terminal, in Computing Skills and the User Interface, Edited by M. J. Coombs and J. L. Alty, Academic Press, London, p. 427-463.

26. M. H. Chignell and P. A. Hancock, Intelligent Interface Design, in *Handbook of Human-Computer Interaction*, edited by M. Helander, Elsevier Science Publishers, Amsterdam, p. 969 – 995.
27. Arisholm, E., Sjoberg, D. I., and Jorgensen, M., 2001. Assessing the Changeability of two Object-Oriented Design Alternatives – a Controlled Experiment, *Empirical Software Engineering Journal*, Volume 6, No. 3, 231-277.
28. Belletini, C., Damiani, E., Fugini, M. G., 2001. Software Reuse In-The-Small: Automating Group Rewarding, *Journal of Information and Software Technology*, Vol. 43, 651-660.
29. Kazman, R., Klein, M., Clements, P., 2000. ATAM: Method for Architecture Evaluation, CMU-SEI Technical Report CMU/SEI-2000-TR-004.
30. Lassing, N., Bengtsson, P., Vliet, Hans van, Bosch, J., 2002. Experiences with ALMA: Architecture-Level Modifiability Analysis, *The Journal of Systems and Software*, Vol. 61, 47-57.
31. Bosch, J., 2000. *Design and Use of Software Architecture*, ACM Press, Harlow, England.
32. Cleal, D. M., and Heaton, N. O., 1988. *Knowledge-Based Systems: Implications for Human-Computer Interfaces*, Ellis Horwood Ltd., England, p. 119.
33. Kolodner, J., 1993, *Case-Based Reasoning*, Morgan Kaufmann Publishers, San Mateo, CA, p. 60
34. Schreiber, G., Akkermans, H., et al., 2000. *Knowledge Engineering and Data Management: The CommonKADS Methodology*, MIT Press, Cambridge, MA.
35. Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M., 1990. Telos: Representing Knowledge About Information Systems, *ACM Transactions on Information Systems*, Vol. 8, No. 4, 325-362.
36. Jarke, M., Jeusfeld, M. A., and Quix, C., 1998. ConceptBase V5.0 User Manual available from <http://www-i5.informatik.rwth-achen.de/Cbdoc>
37. N. Subramanian and L. Chung, “SAAA – A Tool To Develop Adaptable Architectures”, Proceedings of the International Conference on Software Engineering Research and Practice, Las Vegas, June, 2002, CSREA Press, pp. 63-69.